InterSystems Cloud Manager ガイド

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

InterSystems Cloud Manager ガイド
InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

図一覧

テーブル一覧

1

ICM の概要

重要　InterSystems IRIS リリース 2023.3 以降、InterSystems Cloud Manager (ICM) は非推奨になっています。今後のバージョンからは削除される予定です。

このドキュメントでは、InterSystems Cloud Manager (ICM) を使用して、パブリック・クラウドとプライベート・クラウド、および既存の物理クラスタと仮想クラスタに InterSystems IRIS® Data Platform 構成を導入する方法について説明します。

この章では、InterSystems Cloud Manager (ICM) の機能と仕組み、および ICM を使用してクラウド・インフラストラクチャ、仮想インフラストラクチャ、および物理インフラストラクチャに InterSystems IRIS Data Platform 構成を導入する方法について説明します。(実践演習を含む ICM の簡単な紹介については、"InterSystems IRIS デモ : InterSystems Cloud Manager" を参照してください。)

残りの章および付録の内容は、次のとおりです。

- ICM の概要では、ICM の目的、設計、使用方法、および利点の概要について説明します。
- 基本的な ICM の要素では、Docker イメージと Docker リポジトリ、プラットフォーム、構成ファイルなど、ICM の使用に関連する要素について説明します。
- ICM の使用では、ICM を使用してインフラストラクチャをプロビジョニングし、サービスを導入するための詳細な手順について説明します。
- ICM リファレンスでは、コマンド、オプション、および構成パラメータ、ノード・タイプとトポロジ、ライセンスとセキュリティ、マルチゾーン、マルチリージョン、およびプライベート・ネットワーク導入、モニタリング、トラブルシューティングなど、ICM のリファレンス情報を提供します。
- コンテナレスの導入では、ICM を使用して、プロビジョニングしたインフラストラクチャにコンテナ化されていないサービスを導入する方法について説明します。
- ICM 導入環境の共有では、ICM の分散管理モードについて説明します。このモードを使用すると、1 つの導入環境のインフラストラクチャとサービスを複数の ICM コンテナから管理できます。
- ICM によるスクリプト作成では、スクリプトを使用して ICM でインフラストラクチャをプロビジョニングし、サービスを導入する方法について説明します。
- カスタム・コンテナおよびサードパーティ・コンテナでの ICM の使用では、ICM を使用してカスタム・コンテナとサードパーティ・コンテナにサービスを導入するための手順について説明します。
- 既存のクラスタへの導入では、ICM を使用して既存の物理インフラストラクチャまたは仮想インフラストラクチャにサービスを導入するための手順について説明します。

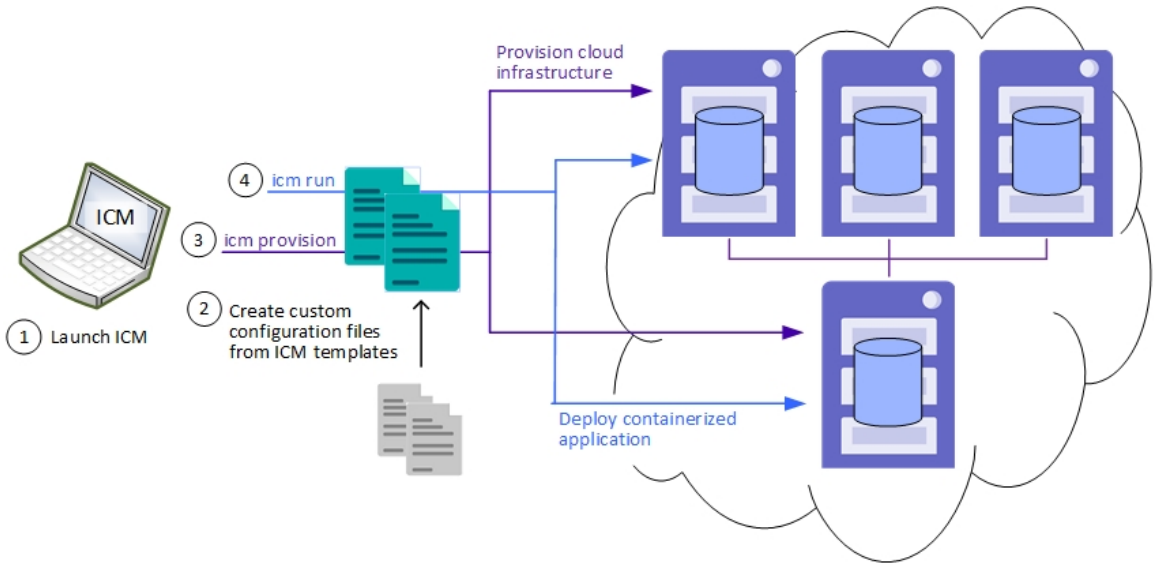
1.1 InterSystems Cloud Manager の利点

InterSystems Cloud Manager (ICM) を使用して、クラウド・インフラストラクチャのプロビジョニングと、インフラストラクチャへのサービスの導入を簡単に直観的な方法で行うことができます。ICM は、Infrastructure as Code (IaC)、不変性を維持するインフラストラクチャ、および InterSystems IRIS をベースとしたアプリケーションのコンテナ化導入の利点を生かせるように設計されており、新しいテクノロジや要員トレーニングへの大規模な投資や、試行錯誤による構成と管理は必要ありません。

ICM により、Google Cloud Platform、Amazon Web Services、Microsoft Azure、Tencent Cloud などの Infrastructure as a Service (IaaS) パブリック・クラウド・プラットフォームで適切な InterSystems IRIS 構成のプロビジョニングと導入が簡単になります。プレーン・テキストの構成ファイルに必要な内容を定義し、簡単なコマンド行インタフェースを使用して ICM に指示を出せば、その他の処理は ICM が行います。例えば、クラウド・インフラストラクチャをプロビジョニングし、InterSystems IRIS ベースのアプリケーションを Docker コンテナ内で導入する処理が行われます。

ICM は、コードのようにチーム・メンバ間で共有し、編集、レビュー、およびバージョン管理を行うことができる宣言型構成ファイルに API を体系化します。ICM を使用すると、これらのファイルで指定された内容を実行することにより、プロダクション・インフラストラクチャを安全かつ予想どおりに作成し、継続的に変更を加えて、改善することができます。

図 1-1: ICM で実現が容易に



ICM を使用すれば、大規模な開発や設備更新を必要とせずに、仮想/クラウド・コンピューティングとコンテナ化されたソフトウェアの提供する効率性、俊敏性、再現性を生かすことができます。ICM は、スタンドアロン・インスタンスから、データ・サーバに接続されたアプリケーション・サーバの分散キャッシュ・クラスタ、さらにデータおよび計算ノードのシャード・クラスタに至るまで、幅広い InterSystems IRIS 構成のプロビジョニングと導入に対応できます。ICM は、既存の仮想クラスタと物理クラスタ、およびプロビジョニングするインフラストラクチャで導入を行うことができます。

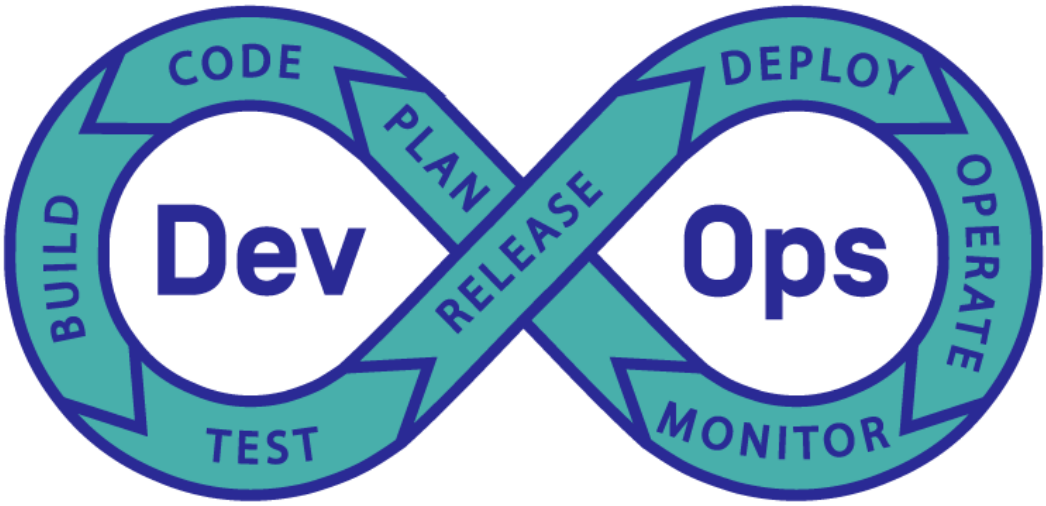
クラウド・インフラストラクチャまたはコンテナ、あるいはその両方をすでに使用している場合でも、ICM はユーザが提供する情報に基づいて多数の手動ステップを自動化するので、アプリケーションのプロビジョニングと導入に必要な時間と労力が大幅に節約されます。また、サードパーティ・ツールの使用と社内のスクリプト作成によって ICM の機能が簡単に拡張され、より高度な自動化と労力の削減が実現されます。

以下のように、ICM アプローチの要素にはそれぞれ固有の利点があり、互いに相乗効果を生み出します。

- 構成ファイル・テンプレートにより、ほとんどの設定についてインターシステムズ提供の既定値をそのまま使用し、必要な値のみをユーザ固有のニーズに合うようにカスタマイズできます。

- コマンド行インタフェースを使用して、プロビジョニングと導入のプロセスの各フェーズを 1 つの単純なコマンドで開始でき、導入したコンテナをさまざまな方法で操作できます。
- IaC により、複製、管理、および破棄が容易で一貫性と再現性のあるプラットフォームを迅速にプロビジョニングできます。
- IaaS プロバイダを利用すれば、最も効率的な方法でインフラストラクチャを活用できます。例えば、クラウド構成が数時間のみ必要な場合は、数時間分のみの料金を支払うだけで済むと共に、再現性がサポートされ、ネットワーキングやセキュリティ、ロード・バランサ、ストレージ・ボリュームなど、ホスト・ノードの実行に必要なリソースがすべて提供されます。
- コンテナ化されたアプリケーションの導入により、不変性を保ったソフトウェア・プロビジョニングによるインフラストラクチャで、シームレスに置換可能なアプリケーション環境が実現します。この環境では、コードがデータから分離されてインフラストラクチャ自体を更新するリスクとコストが回避されると共に、継続的統合/継続的導入(CI/CD)とDevOpsアプローチがサポートされます。

図 1-2: DevOps アプローチをサポートするコンテナ



これらの優れた機能を活用する ICM の利点は、以下のとおりです。

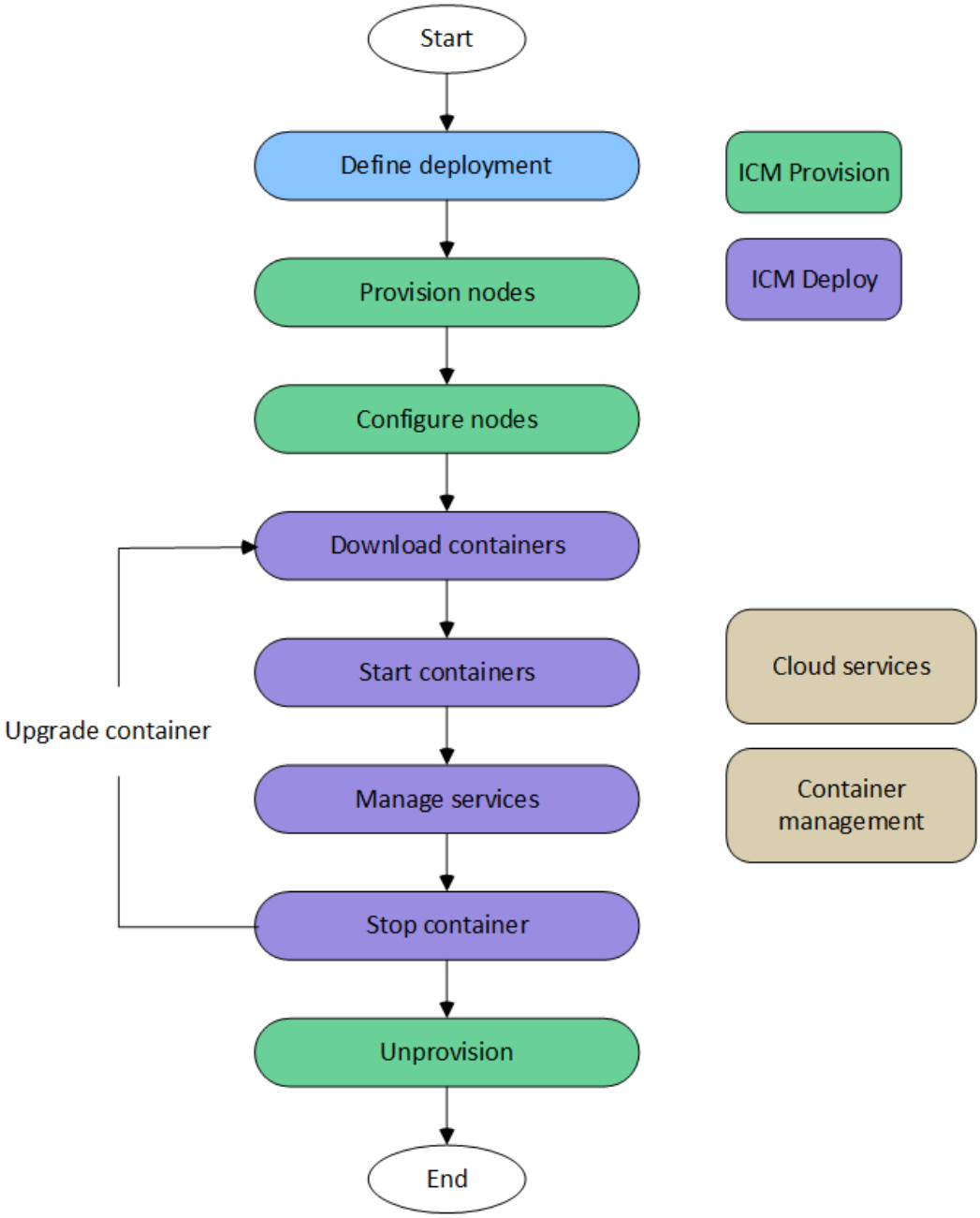
- 大規模でクラウド・ベースの InterSystems IRIS 構成のプロビジョニングと導入が自動化され、コマンド行での管理が実現されます。
- 既存の InterSystems IRIS および InterSystems IRIS ベースのアプリケーションが、企業の DevOps ツールチェーンと統合されます。
- アプリケーションと実行環境の両方のバージョン管理が容易で、安定性と堅牢性がありリスクが最小化されます。
- 迅速な再プロビジョニングと再導入により、導入済みの InterSystems IRIS 構成を弾力的に拡張/縮小できます。

Docker コンテナを使用しない場合は、ICM を使用してクラウド・インフラストラクチャをプロビジョニングし、コンテナ化されていない InterSystems IRIS インスタンスをそのインフラストラクチャにインストールするか、InterSystems IRIS を既存のインフラストラクチャにインストールすることができます。ICM のコンテナレス・モードの使用の詳細は、付録 "コンテナレスの導入" を参照してください。

1.2 InterSystems Cloud Manager アプリケーションのライフサイクル

プロビジョニングと導入の 2 つの主なフェーズを含む、アプリケーション・ライフサイクルで ICM が担当するロールを以下の図に示します。

図 1-3: アプリケーション・ライフサイクルでの ICM のロール



1.2.1 目標の定義

ICM の構成ファイルには、出荷時の状態で、必要な InterSystems IRIS 構成のプロビジョニングと導入のために必要な設定のほとんどすべてが含まれています。適切なファイルで目的の構成を定義すると共に、資格情報 (クラウド・サーバ・プロバイダ、SSH、TLS)、InterSystems IRIS ライセンス、必要なホスト・ノードのタイプとサイズなど、いくつかの詳細を指定すれば済みます。(詳細は、"導入の定義" を参照してください。)

注釈　このドキュメントでは、ホスト・ノードという用語は、クラウドでプロビジョニングされる仮想ホストを指します。クラウドは、サポートされるクラウド・サービス・プロバイダのパブリック・クラウド、または VMware vSphere を使用するプライベート・クラウドになります。

1.2.2 プロビジョニング

ICM は、4 つの主なプロビジョニング処理をサポートします。これらの処理は、クラウド環境内のホスト・ノードおよび関連リソースの作成 (プロビジョニング)、構成、変更、および破棄 (プロビジョニング解除) です。

ICM がプロビジョニング・タスクを実行する際には、HashiCorp の Terraform を呼び出します。Terraform は、インフラストラクチャの作成、変更、およびバージョン管理を安全、効率的に行うためのオープン・ソース・ツールで、既存のクラウド・サービス・プロバイダとカスタム・ソリューションの両方と互換性があります。構成ファイルは、プロビジョニングされるインフラストラクチャを記述します。(詳細は、"インフラストラクチャのプロビジョニング" を参照してください。)

すべてのタスクを個別の Terraform コマンドとして発行できますが、ICM によって Terraform ジョブを実行すると、Terraform を直接呼び出す場合と比べて以下の利点があります。

Terraform を直接実行する場合	ICM によって Terraform を実行する場合
プロビジョニング・タスクのみを実行し、プロビジョニングを導入や構成と統合することはできない	弾力的な再プロビジョニングおよび再導入のフェーズも含め、すべてのフェーズを統合 (例えば、クラスタ・インフラストラクチャにノードを追加した後、各ノードで InterSystems IRIS を導入および構成して、それらをクラスタに組み込む)
それぞれのタイプのノードを順次構成するので、長いプロビジョニング時間がかかる	複数の Terraform ジョブを並列に実行してすべてのノード・タイプを同時に構成するので、プロビジョニングが迅速
プログラムによるアクセスは不可能 (API がない)	Terraform へのプログラムによるアクセスが可能
専有の HashiCorp Configuration Language (HCL) で目的のインフラストラクチャを定義	汎用の JSON 形式で目的のインフラストラクチャを定義

また ICM は、SSH を同じ方法で使用してプロビジョニング後の構成タスクを実行します。つまり、複数のノードでコマンドを並列に実行して、実行を高速化します。

1.2.3 導入

ICM は、Docker コンテナに入った InterSystems IRIS イメージを、プロビジョニング対象のホスト・ノードに導入します。これらのコンテナは、プラットフォームから独立し、完全にポータブルなので、インストールは必要なく、調整は簡単です。ICM 自体は Docker コンテナ内で導入されます。コンテナ化されたアプリケーションは、ホスト・システムのカーネルでネイティブに実行されます。一方、アプリケーション実行に必要な要素と、必要な接続、サービス、およびインタフェースにアクセス可能にするために必要な要素 (実行時環境、コード、ライブラリ、環境変数、および構成ファイル) のみが、コンテナで提供されます。

導入タスクは、Docker を呼び出すことによって実行されます。すべてのタスクを個別の Docker コマンドとして発行できますが、ICM によって Docker コマンドを実行すると、Docker を直接呼び出す場合と比べて以下の利点があります。

- ICM は Docker コマンドをすべてのマシンにわたって並列スレッドで実行するので、イメージのプル (ダウンロード) など、長時間かかるタスクを実行するための合計時間が短縮されます。
- ICM は、アプリケーション固有の要件があるタスク (ローリング・アップグレードなど) を調整できます。
- ICM は、最初の導入以降に変更された (コンテナのアップグレードや新しいコンテナの追加を含む) インフラストラクチャにサービスを再導入できます。

注釈　コマンド行で InterSystems IRIS コンテナをすばやく実行する方法は、"InterSystems IRIS の基礎：InterSystems IRIS コンテナの実行" を参照してください。ICM 以外の方法を使用してコンテナに InterSystems IRIS および InterSystems IRIS ベースのアプリケーションを導入する方法は、"コンテナ内でのインターシステムズ製品の実行" を参照してください。

1.2.4 管理

ICM コマンドを使用して、インフラストラクチャとコンテナをさまざまな方法で操作および管理できます。例えば、クラウド・ホスト上またはコンテナ内でコマンドを実行したり、ホストまたはコンテナにファイルをコピーしたり、コンテナをアップグレードしたり、InterSystems IRIS を直接操作したりすることができます。

ICM サービスの導入と管理の詳細は、"サービスの導入と管理" を参照してください。

1.3 InterSystems IRIS のその他の自動導入方法

InterSystems IRIS データ・プラットフォームには、ICM のほかに、以下の自動導入方法が用意されています。

1.3.1 InterSystems Kubernetes Operator (IKO) を使用した自動導入

Kubernetes は、コンテナ化されたワークロードとサービスの導入、拡張、および管理を自動化するためのオープンソースのオーケストレーション・エンジンです。導入するコンテナ化されたサービスと、そのサービスを管理するポリシーを定義すると、Kubernetes は、必要なリソースを可能な限り最も効率的な方法で透過的に提供します。また、導入が指定値から外れた場合は導入を修復またはリストアするほか、拡張を自動またはオンデマンドで行います。InterSystems Kubernetes Operator (IKO) は、IrisCluster カスタム・リソースで Kubernetes API を拡張します。このリソースは、InterSystems IRIS のシャード・クラスタ、分散キャッシュ・クラスタ、またはスタンドアロン・インスタンスとして、すべて任意でミラーリングした状態で、Kubernetes プラットフォームに導入できます。

Kubernetes で InterSystems IRIS を導入するのに IKO は必須ではありませんが、プロセスが大幅に簡易化され、InterSystems IRIS 固有のクラスタ管理機能が Kubernetes に追加され、クラスタにノードを追加するなどのタスクが可能になります。このようなタスクは、IKO を使わなければインスタンスを直接操作して手動で行わなければなりません。

IKO の使用法の詳細は、"InterSystems Kubernetes Operator の使用" を参照してください。

1.3.2 構成マージを使用した自動導入

Linux および UNIX® システムで利用可能な構成マージ機能を使用すると、宣言型構成マージ・ファイルを導入内の各インスタンスに適用するだけで、同じイメージから導入した InterSystems IRIS コンテナや、同じキットからインストールしたローカル・インスタンスの構成を変更することができます。

このマージ・ファイルは、既存のインスタンスを再起動したときに適用することもでき、インスタンスの構成パラメータ・ファイル (CPF) を更新します。CPF にはインスタンスのほとんどの構成設定が含まれており、これらの設定は、インスタンス導入後の最初の起動を含め、開始時に毎回、CPF から読み取られます。導入時に構成マージを適用すると、インスタンスと共に提供された既定の CPF が実質的に独自の更新バージョンに置き換えられます。

構成マージを使用すると、希望どおりの構成で、個々のインスタンスまたはインスタンスのグループを導入できます。複雑なアーキテクチャを導入する場合は、導入するインスタンスのタイプごとに、別個のマージ・ファイルを呼び出すことができます。例えば、シャード・クラスタを導入する場合は、データ・ノード 1、残りのデータ・ノード、計算ノード (オプション) の順に導入します。

“ICM リファレンス” の章の ″カスタマイズされた InterSystems IRIS 構成を使用した導入″ で説明しているように、**UserCPF** パラメータを使用して、ICM 導入フェーズ中に適用する構成マージ・ファイルを指定できます。前のセクションで説明したように、IKO にも構成マージ機能が組み込まれています。

一般的な構成マージの使用法および具体的なミラーの導入方法の詳細は、″構成マージを使用した InterSystems IRIS の自動構成″ を参照してください。

2

基本的な ICM の要素

重要　InterSystems IRIS リリース 2023.3 以降、InterSystems Cloud Manager (ICM) は非推奨になっています。今後のバージョンからは削除される予定です。

この章では、ICM の使用に関係がある、基本的な要素について説明します。

2.1 InterSystems Cloud Manager イメージ

ICM は Docker イメージとして提供されます。これを実行すると、ICM コンテナが起動されます。例えば Terraform、Docker クライアント、構成ファイルのテンプレートなど、プロビジョニング、導入、および管理のタスクを実行するために ICM が必要とするすべてがこのコンテナに含まれています。ICM コンテナについての詳細は、"ICM の起動" を参照してください。

ICM を起動するシステムは、Docker コンテナをホストするプラットフォームとして Docker によってサポートされている必要があり、Docker がインストール済みで、インフラストラクチャのプロビジョニングとコンテナの導入を行うプロビジョニング・プラットフォーム、または導入を行う既存のインフラストラクチャへの十分な接続性を備えていることが必要です。

2.2 プロビジョニング・プラットフォーム

ICM は、仮想ホスト・ノードと関連リソースを以下のプラットフォームにプロビジョニングできます。

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure (Azure)
- Tencent Cloud (Tencent)
- VMware vSphere (vSphere)

  注釈　VMware vSphere を利用している多数のユーザのニーズに対処するために、このリリースの ICM では、vSphere がサポートされています。特有の vSphere 構成と基盤のハードウェア・プラットフォームに応じて、ICM を使用して仮想マシンをプロビジョニングするには、このガイドに記載されていない追加の拡張や調整が必要になることがあります。大規模で複雑な導入の場合は特に、その可能性があります。また、ICM を使用した仮想マシンのプロビジョニングがプロダクションでの使用に適さないこともあります。完全なサポートは、今後のリリースで予定されています。

AWS、GCP、Azure、および Tencent では、ICM は、1 つのリージョン内の複数のゾーン、複数のリージョン、さらにはさまざまなクラウド・プロバイダ・プラットフォームにわたって 1 つの構成をプロビジョニングして導入することができます。

これらのいずれかのプラットフォームで ICM を使用する前に、プラットフォームに関する全般的な知識が必要です。また、アカウントの資格情報も必要です。詳細は、"セキュリティ関連ファイルの入手" を参照してください。

ICM は、既存の仮想クラスタまたは物理クラスタ (プロバイダ PreExisting) を必要に応じて構成し、これらのクラスタにコンテナを導入することもできます (自身がプロビジョニングするノードの場合と同様に)。

2.3 導入プラットフォーム

インターシステムズのコンテナ・イメージは Open Container Initiative (OCI) の仕様に準拠しており、Docker Enterprise Edition エンジンを使用して構築されます。このエンジンは、OCI 標準を全面的にサポートしており、これによってイメージは認定され、Docker Hub レジストリで公開することができます。

インターシステムズのイメージは、広く知られたコンテナ向け Ubuntu オペレーティング・システムと ICM を使用して構築およびテストされているため、オンプレミスとパブリック・クラウドの両方において、Linux ベースのオペレーティング・システム上の OCI に準拠するすべてのランタイム・エンジンで導入がサポートされます。

2.4 導入に含めるノードの定義

プロビジョニング対象ノードごとに 1 つの InterSystems IRIS インスタンスが ICM によって導入され、各インスタンスが InterSystems IRIS 構成で担当するロールは Role フィールドの値によって決まります。ノードとそのインスタンスは、そのロールの下でプロビジョニング、導入、および構成されます。

ICM の使用準備の際に、構成に含めるノードのタイプと数を選択することによってターゲット構成を定義する必要があります (“ICM の使用” の章にある "導入の定義" を参照)。例えば、InterSystems IRIS シャード・クラスタを導入する場合は、クラスタに含めるデータ・ノードおよび (オプションで) 計算ノードの数と、データ・ノードをミラーリングするかどうかを事前に決定しておく必要があります。その後、ICM に必要な入力を提供するために、対象のノードの仕様を定義ファイルに入力します ("構成ファイル、状態ファイル、およびログ・ファイル" を参照)。以下の単純化した例にこれを示します。ここでは、4 つのデータ・ノードと 8 つの計算ノードを定義します。

```
[
    {
        "Role": "DATA",
        "Count": "4",
        "LicenseKey": "ubuntu-sharding-iris.key"
    },
    {
        "Role": "COMPUTE",
        "Count": "8",
        "StartCount": "5",
        "LicenseKey": "ubuntu-sharding-iris.key"
    }
]
```

データ・ノードをミラーリングするかどうかを決定する Mirror フィールドは、既定値ファイルで指定します。ミラー導入については、"ICM クラスタのトポロジとミラーリング" で詳しく説明します。

2.5 フィールド値

ICM によって行われるプロビジョニング、導入、および構成の作業は、ユーザが指定するいくつかのフィールド値によって決まります。例えば、Provider フィールドは、使用するプロビジョニング・プラットフォームを指定します。値が AWS ならば、指定されたノードは AWS 上でプロビジョニングされます。

フィールド値を指定するには、以下の 2 つの方法があります。

- 一部はコマンド行で指定できます ("コマンド行" を参照)。
- すべてを 2 つの構成ファイルに含めることができますが ("構成ファイル、状態ファイル、およびログ・ファイル" を参照)、defaults.json ファイルのみで指定できるものもあれば、定義ファイルのみで使用されるものもあります。

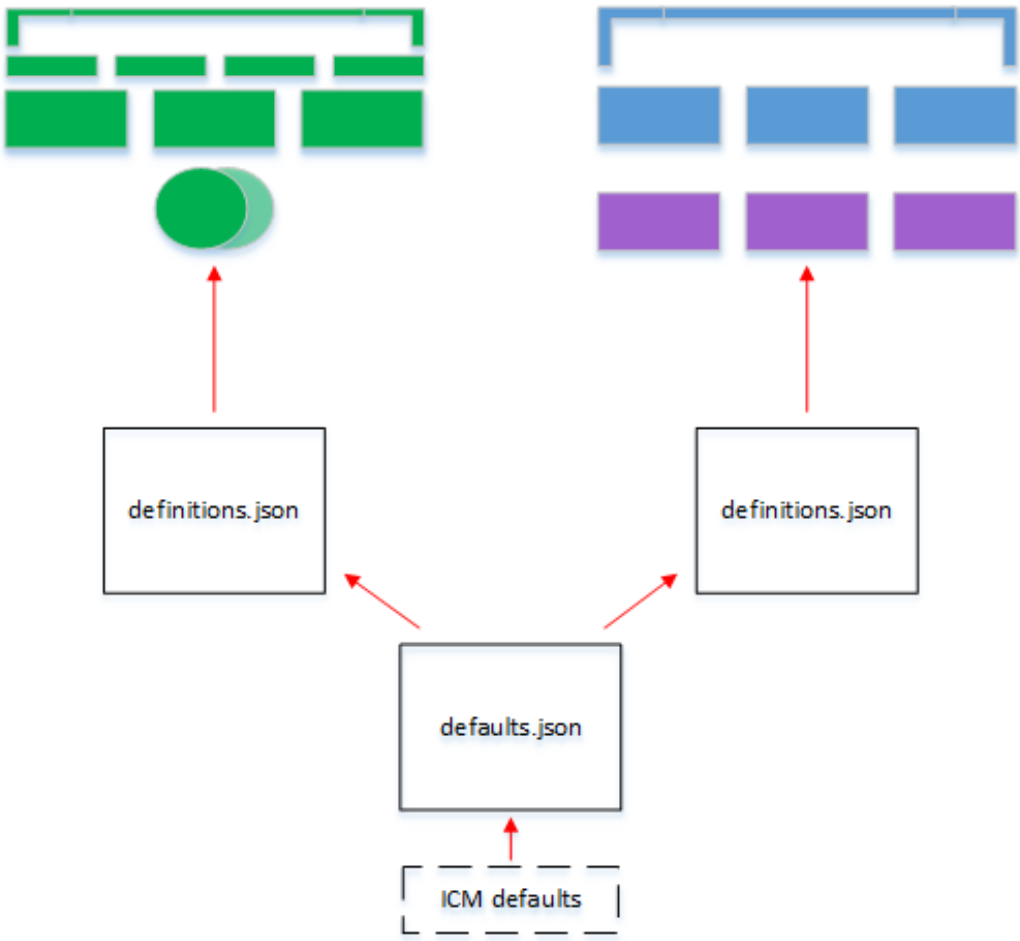
ほとんどの ICM フィールドには既定値もあります。優先度が高い順に、これらの指定の順位は以下のとおりです。

1. コマンド行オプション
2. definitions.json 構成ファイルのエントリ
3. defaults.json 構成ファイルのエントリ
4. ICM の既定値

これにより、よく使用されるフィールドを繰り返し指定することなく、柔軟な指定が可能です。既定値ファイル (defaults.json) を使用して、特定のカテゴリに属する複数の導入に値を指定できます (値が必須の場合、または既定値をオーバーライドする場合)。例えば、同じプラットフォーム上でプロビジョニングされる導入のためにこのファイルを使用できます。定義ファイル (definitions.json) は、クラスタ内のノードとそれを識別するラベルを指定するなど、特定の導入に対して値を指定します。ただし、特定の導入で特定のノード・タイプの defaults.json 値をオーバーライドするために、既定値ファイルに含まれるフィールドを指定することもできます。コマンド行オプションとしてフィールド値を指定すると、特定の導入で特定のコマンドのコンテキストにおいて両方の構成ファイルをオーバーライドできます。

以下の図は、1 つの defaults.json ファイルを共有し、異なる definitions.json ファイル (1 つ目は分散キャッシュ・クラスタのもので、2 つ目はシャード・クラスタのもの) を使用する 2 つの導入を示しています。

図 2-1: 導入を定義する ICM 構成ファイル



これらのファイルで指定できる必須のフィールドとオプションのフィールドの包括的なリストについては、“ICM リファレンス” の章にある ″ICM の構成パラメータ″ を参照してください。

2.6 コマンド行

ICM コマンド行インタフェースにより、ユーザは調整プロセスのアクションを指定できます (例えば、インフラストラクチャをプロビジョニングする icm provision や、指定したコンテナを実行して導入を行う icm run)。また、コンテナのアップグレードや InterSystems IRIS への接続など、コンテナやサービスの管理コマンドを指定することもできます。ICM は、構成ファイルからの情報を使用してこれらのコマンドを実行します。入力構成ファイルのどちらか (″構成ファイル、状態ファイル、およびログ・ファイル″ を参照) がコマンド行で指定されない場合、ICM は現在の作業ディレクトリに存在するファイル (**definitions.json** または **defaults.json**) を使用します。一方または両方のファイルが指定されず、存在しない場合、コマンドは失敗します。

コマンド行を使用して、オプションを指定することもできます。以下のように、オプションにはいくつかの目的があります。

- 単一のコマンドのみを対象に、構成ファイル内の情報をオーバーライドするため。例えば、以下に示すように、構成ファイルに指定されているものとは別の Docker イメージを導入するため。

```
icm run -image image
```

- 情報が維持されることを避ける理由で構成ファイルに含めない情報を指定するため。例えば、以下のようにパスワードを指定します。

  ```
  icm run -iscPassword password
  ```

- 導入内の 1 つ以上のノード上で実行されるコマンドに関連した情報を指定するため。例えば、以下のようにクエリを指定します。

  ```
  icm sql -role DATA -namespace namespace -command "query"
  ```

オプションが異なるコマンドで使用される場合、オプションの目的は異なることがあります。例えば、icm run コマンドでは、–container オプションは指定イメージから起動される新規コンテナの名前を指定しますが、icm ps コマンドでは、状態情報を表示する既存の導入済みコンテナの名前を指定します。

ICM コマンドとコマンド行オプションの総合的なリストについては、"ICM コマンドおよびオプション" を参照してください。

2.7 構成ファイル、状態ファイル、およびログ・ファイル

ICM は、以下のように JSON ファイルを入力と出力の両方として使用します。

- 入力としては、構成ファイルでは ICM でタスクの実行に必要な情報を提供します (使用するプロビジョニング・プラットフォームや、プロビジョニングするノードのタイプと数など)。これらのファイルはユーザによって提供されます。これらは ICM に付属するテンプレートから変更して作成することも、手動で作成することも、スクリプトまたは UI の出力によって作成することもできます。
- 出力としては、ICM によって生成されるファイルは ICM のタスクの結果を記述します。

ICM タスクの結果、新規または変更されたデータ (例えば、IP アドレス) が生成される場合、その情報は後続のタスクが使用するために JSON 形式で記録されます。ICM では、認識されないフィールドや、現在のタスクに適用されないフィールドをすべて無視しますが、これらのフィールドを後続のタスクに渡し、エラーは生成しません。この動作には、以下の利点があります。

- 前のフェーズ (プロビジョニングなど) で指定された情報が後のフェーズ (導入など) で使用され、複数の構成ファイルを編集したり保守したりする必要がありません。
- 複数の ICM バージョン間で一定の前方互換性と後方互換性がサポートされます。
- 複数のプロバイダに対して 1 つの構成ファイルを使用するために必要な作業が最小限になります。
- JSON 標準では、コンテンツをコメント化する手段は正式には提供されていませんが、名前を変更することによってフィールドを "コメント化" できます。

ICM によって使用される JSON ファイルには、以下のものがあります。

- 定義ファイルと既定値ファイル。これは、ICM のプロビジョニング・フェーズと導入フェーズへの入力としてユーザによって提供されます。既定では、これらのファイルは現在の作業ディレクトリにあることが想定されます (これらのファイルで指定できる必須のフィールドとオプションのフィールドの包括的なリストについては、"ICM リファレンス" の章にある "ICM の構成パラメータ" を参照してください)。
- インスタンス・ファイル。これは、プロビジョニング・フェーズの最後に ICM によって生成され、導入フェーズへの入力として使用されます。既定では、このファイルは現在の作業ディレクトリに作成されます。
- Terraform 状態ファイル。これは、プロビジョニング・フェーズ中に ICM によって生成され、プロビジョニング解除など、以後のインフラストラクチャ関連の操作への入力として使用されます。既定では、これらのファイルは、現在の作業ディレクトリの下に ICM によって生成される状態ディレクトリに配置されます。

ICM はほかにも、いくつかのログ・ファイル、出力ファイル、およびエラー・ファイルを生成します。これらは、現在の作業ディレクトリ、または状態ディレクトリに配置されます。

2.7.1 定義ファイル

定義ファイル (definitions.json) は、特定の導入のためにプロビジョニングされるホスト・ノードのセットを記述します。ICM は、現在の作業ディレクトリで見つかった定義ファイルを使用します。

この定義ファイルは、ノード定義を表す一連の JSON オブジェクトから成ります。それぞれのオブジェクトは、属性のリスト、およびそのタイプのノードをいくつ作成する必要があるかを指示するカウントを含みます。必須のフィールドとオプションのフィールドがあり、一部のフィールドはプロバイダ固有です (つまり、AWS、Google Cloud Platform、Microsoft Azure、Tencent、VMware vSphere、または PreExisting で使用)。

このファイルにはほとんどのフィールドを入力でき、ノード定義ごとに必要なだけ繰り返すことができます。ただし、フィールドによっては、1 つの導入された構成全体で一致している必要があるので、定義ファイルのエントリによって既定値から変更することはできません。また、既定値がない場合に、フィールドを指定することもできません。Provider フィールドは分かりやすい例です (明らかな理由から)。definitions.json ファイルに含めることができないその他のフィールド (含めるとエラーが発生します) は、Label および Tag です。

ノード・タイプ (Role など) によって異なるフィールドは、definitions.json のノード定義に含める必要があります。definitions.json は、特定のノード・タイプについて ICM の既定値または defaults.json ファイルの設定をオーバーライドする場合にも使用されます。例えば、以下の例は、AWS 上のミラーリングされたデータ・サーバ ("Role": "DM")、3 つのアプリケーション・サーバ ("Role": "AM")、およびミラー・アービター・ノード ("Role": "AR") で構成される分散キャッシュ・クラスタをプロビジョニングするためのサンプル definitions.json ファイルの内容を示しています。

- DataVolumeSize フィールドは、DM ノードについてのみ指定されています。他のノードは ICM の既定値を使用するからです。
- DM ノードおよび AR ノードの定義には、defaults.json で指定されている既定のインスタンス・タイプをオーバーライドする InstanceType フィールドが含まれています。
- AR ノードの定義には、defaults.json 内の該当フィールドをオーバーライドする DockerImage フィールドが含まれています。AR ノードには InterSystems IRIS コンテナではなく、アービター・コンテナが導入されるからです。

一部のフィールドは特定のノード・タイプに限定されているので、definitions.json で指定する必要があります。例えば、この AM ノードの定義には、AM ノードについてロード・バランサを自動的にプロビジョニングする "LoadBalancer": "true" が含まれています。この設定は WS ノードでも使用できますが、他のノード・タイプに適用するとエラーが発生します。

```
[
    {
        "Role": "DM",
        "Count": "2",
        "DataVolumeSize": "50",
        "InstanceType": "m4.xlarge"
    },
    {
        "Role": "AM",
        "Count": "3",
        "StartCount": "3",
        "LoadBalancer": "true"
    },
    {
        "Role": "AR",
        "Count": "1",
        "StartCount": "6",
        "InstanceType": "t2.small",
        "DockerImage": "intersystems/arbiter:latest-em"
    }
]
```

definitions.json ファイルを変更した後、再プロビジョニングや再導入を行うことによって、ノードを追加または削除したり、既存のノードを変更したりして、既存の構成を弾力的に拡張/縮小および変更することができます。

注釈　このドキュメントで示しているイメージ・タグは、例として紹介しているだけです。InterSystems Container Registry (ICR) に移動して、最新のリポジトリとタグを参照してください。

2.7.2 既定値ファイル

一般に、既定値ファイルは特定のタイプの導入すべてにわたって同一フィールドを定義します (特定のクラウド・プラットフォームでプロビジョニングされるものなど)。ICM は、現在の作業ディレクトリで見つかった既定値ファイルを使用します。

"定義ファイル" で説明したように、ほとんどのフィールドはどちらの入力ファイルにも指定できますが、一部は導入全体にわたって一致している必要があり、ノード・タイプごとに別個に指定することはできません (例えば、Provider)。これらのフィールドのほかに、場合によってはすべての導入にわたってすべてのノードにフィールドを適用し、必要に応じてコマンド行または定義ファイルでこれらをオーバーライドしたいこともあります。両方のタイプのフィールドが defaults.json ファイルに含まれています。既定値ファイルに可能な限り多くのフィールドを含めることにより、定義ファイルが小さく保たれ、管理しやすくなります。

既定値ファイルの形式は、単一の JSON オブジェクトです。このファイルに含まれている値は、コマンド行オプションまたは定義ファイルで値が指定されていない (または null である) すべてのフィールドに適用されます。

"定義ファイル" で示した definitions.json ファイルと共に使用されるサンプル defaults.json ファイルの内容を以下に示します。OSVolumeSize、DataVolumeSize、InstanceType など、既定値ファイルで指定される既定値の一部は定義ファイルによってオーバーライドされます。

```
{
    "Provider": "AWS",
    "Label": "Sample",
    "Tag": "TEST",
    "DataVolumeSize": "10",
    "SSHUser": "ubuntu",
    "SSHPublicKey": "/Samples/ssh/insecure-ssh2.pub",
    "SSHPrivateKey": "/Samples/ssh/insecure",
    "DockerRegistry": "https://containers.intersystems.com",
    "DockerImage": "containers.intersytems.com/intersystems/iris:some-tag",
    "DockerUsername": "xxxxxxxxxxxx",
    "DockerPassword": "xxxxxxxxxxxx",
    "TLSKeyDir": "/Samples/tls/",
    "LicenseDir": "/Samples/license/",
    "Region": "us-east-1",
    "Zone": "us-east-1a",
    "AMI": "ami-07267eded9a267f32",
    "DockerVersion": "5:20.10.17~3-0~ubuntu-jammy",
    "InstanceType": "m5.large",
    "Credentials": "/Samples/AWS/sample.credentials",
    "ISCPassword": "",
    "Mirror": "false",
    "UserCPF": "/Samples/cpf/iris.cpf"
}
```

2.7.3 インスタンス・ファイル

インスタンス・ファイル (instances.json) は、プロビジョニング・フェーズ中に ICM によって生成され、正常にプロビジョニングされたホスト・ノードのセットを記述します。この情報は導入および管理フェーズへの入力となるので、このフェーズ中にファイルが使用可能であることが必要で、ファイルが現在の作業ディレクトリにない場合はパスを ICM に対して指定する必要があります。インスタンス・ファイルは現在の作業ディレクトリに作成されます。

定義ファイルにはノード・タイプごとにただ 1 つのエントリが含まれ、各エントリにはそのタイプのノード数を指定する Count 値が含まれていますが、インスタンス・ファイルには実際にプロビジョニングされるノードごとにエントリがあります。例えば、前記のサンプル定義ファイルには 3 つのエントリがありますが (3 つのアプリケーション・サーバ用に 1 つ、2 つのデータ・サーバ用に 1 つ、アービター用に 1 つ)、作成されるインスタンス・ファイルにはプロビジョニングされるノードごとに 1 つずつ、6 つのオブジェクトが含まれます。

各ノードの定義を構成するすべてのパラメータ (定義ファイルと既定値ファイル内のパラメータ、既定値のある構成ファイルで指定されていないパラメータ、内部 ICM パラメータなど) がエントリに含まれるほか、ノードのマシン名 (Label、Role、

および Tag の各フィールドから構成される)、ノードの IP アドレス、および DNS 名も含まれます。導入環境の状態ディレクトリ内にあるそのノードのサブディレクトリの場所も含まれます。

2.7.4 状態ディレクトリと状態ファイル

ICM は、プロビジョニングされたインフラストラクチャの存続中に使用されるいくつかの状態ファイル (ログ、生成されるスクリプトなど) を state サブディレクトリに書き込みます。state サブディレクトリは、既定で現在の作業ディレクトリ内に作成されます。

プロビジョニング時に生成される状態ファイルには、Terraform オーバーライド・ファイルと状態ファイル (terraform.tfvars と terraform.tfstate) のほか、Terraform 出力ファイル、エラー・ファイル、およびログ・ファイルが含まれます。これらの Terraform に関連した一連のファイルは、定義ファイル内のノード・タイプ定義ごとに別々のサブディレクトリ内に作成されます。例を以下に示します。

```
./state/Acme-DM-TEST
./state/Acme-AM-TEST
./state/Acme-AR-TEST
```

独自の状態ディレクトリを別の名前で作成する場合や別の場所に作成する場合は、icm provision コマンドで -stateDir コマンド行オプションを使用して既定の場所をオーバーライドできますが、その後も -stateDir オプションを引き続き使用して、インフラストラクチャをプロビジョニング解除する場合など、後続のすべてのプロビジョニング・コマンドでその場所を指定する必要があります。

重要　ICM は、作成する状態ファイルを使用して、プロビジョニングしたインフラストラクチャに関する正確な最新情報を取得します。これらのファイルがなければ、ICM がプロビジョニング状態を再構成することが困難または不可能になり、エラーが発生するほか、場合によっては手動での介入が必要になります。この理由から、状態ディレクトリは信頼性が高く確実にアクセスできるストレージに配置し、適切なバックアップ・メカニズムを実行することをインターシステムズは強くお勧めします。

2.7.5 ログ・ファイルおよび InterSystems Cloud Manager のその他のファイル

ICM は、いくつかのログ・ファイル、出力ファイル、およびエラー・ファイルを現在の作業ディレクトリと state ディレクトリ・ツリー内に書き込みます。現在の作業ディレクトリ内の icm.log ファイルは、ICM の情報メッセージ、警告メッセージ、およびエラー・メッセージを記録します。state ディレクトリ・ツリー内にあるその他のファイルは、コマンドの結果 (エラーを含む) を記録します。例えば、プロビジョニング・フェーズ中のエラーは、通常は terraform.err ファイルに記録されます。

重要　ICM の操作中にエラーが発生した場合、ICM はログ・ファイルを示すメッセージを表示します。このログ・ファイルで、エラーに関する情報を確認できます。ICM の導入を開始する前に、ログ・ファイルとその場所をよく理解しておいてください。

2.8 Docker リポジトリ

ICM によって導入されるそれぞれのイメージは、Docker リポジトリからプル (ダウンロード) されます。多くの Docker イメージはパブリック Docker リポジトリから自由にダウンロードできます。ただし、インターシステムズのリポジトリなど、プライベート・リポジトリにアクセスするには Docker ログインが必要です。

2.8.1 Docker リポジトリへのログイン

導入フェーズの一環として、ICM はユーザが提供する資格情報を使用して、ユーザが指定する Docker リポジトリに各ノードをログインさせます。これは、構成ファイルのいずれかの DockerImage フィールドによって指定されたイメージ、ま

たは -image option を使用してコマンド行で指定されたイメージを導入する前に行われます。(リポジトリ名がイメージの指定に含まれている必要があります。)以下の 3 つのフィールドを defaults.json ファイルに組み込んで、必要な情報を指定できます。

- DockerRegistry

  DockerImage によって指定されるイメージを保管する Docker リポジトリのホスト・サーバの DNS 名。このフィールドが含まれていない場合、ICM は docker.com にある Docker のパブリック・レジストリを使用します。

  DockerImage によって指定されるリポジトリが、DockerRegistry によって指定されるサーバに存在しない場合、導入は失敗してエラーが返されます。

  InterSystems Container Registry (ICR) の使用の詳細は、“ICM の使用” の章の ″ICM イメージのダウンロード″ を参照してください。

- DockerUsername

  Docker ログインに使用されるユーザ名。パブリック・リポジトリの場合は必要ありません。このフィールドが含まれておらず、DockerImage によって指定されるリポジトリがプライベートの場合、ログインは失敗します。

- DockerPassword

  Docker ログインに使用されるパスワード。パブリック・リポジトリの場合は必要ありません。このフィールドが含まれておらず、DockerImage によって指定されるリポジトリがプライベートの場合、パスワードを入力するように求められます (入力内容はマスクされます)。

  注釈　$、|、(、) などの特殊文字が DockerPassword フィールドの値に含まれる場合は、2 つの ¥ 文字でエスケープする必要があります。例えば、パスワード abc$def は、abc¥¥$def として指定する必要があります。

2.8.2 Docker リポジトリの設定

重要なアプリケーションを実行するためにネットワーク可用性に依存せず、インターシステムズのイメージ (およびユーザ独自のイメージ) をローカルに保管できるように Docker リポジトリを設定できます。この方法については、Docker ドキュメンテーションの ″Deploy a registry server″ を参照してください。

3

InterSystems Cloud Manager の使用

重要　　InterSystems IRIS リリース 2023.3 以降、InterSystems Cloud Manager (ICM) は非推奨になっています。今後のバージョンからは削除される予定です。

この章では、ICM を使用して InterSystems IRIS 構成をパブリック・クラウドに導入する方法について、以下のように説明します。各セクションでは、ICM を使用してサンプル InterSystems IRIS 構成を AWS に導入するために必要な手順について、以下のように説明します。

- この章全体で例として使用される 2 つのサンプル・ユース・ケースに基づいて、導入について検討します。
- インターシステムズ提供の ICM イメージに対して docker run コマンドを使用して、ICM コンテナを起動し、コマンド行を開きます。
- セキュリティ保護された通信を ICM が行うために必要なクラウド・プロバイダと TLS 資格情報を入手します。
- プロビジョニングする各ノード・タイプのノード数と、その他の必要な選択を行った後、導入に必要な構成ファイル **defaults.json** および **definitions.json** を作成します。
- icm provision コマンドを使用してインフラストラクチャをプロビジョニングし、ICM のインフラストラクチャ管理コマンドを検討します。いつでも再プロビジョニングを行って、既存のインフラストラクチャを変更できます (拡張や縮小を含む)。
- icm run コマンドを実行してコンテナを導入し、ICM のコンテナおよびサービスの管理コマンドを検討します。インフラストラクチャが再プロビジョニングされた場合や、コンテナを追加、削除、またはアップグレードする場合は、再導入を行うことができます。
- icm unprovision コマンドを実行して導入を破棄します。

以下のセクションで詳しく説明されている ICM コマンドとコマンド行オプションの総合的なリストについては、"ICM コマンドおよびオプション" を参照してください。

3.1 ICM のユース・ケース

この章では、以下の 2 つの InterSystems IRIS 構成を導入する、2 つの代表的な ICM のユース・ケースを中心に説明します。

- 分散キャッシュ・クラスター ミラーリングされたデータ・サーバ、3 つのアプリケーション・サーバ、アービター・ノード、およびロード・バランサ。この導入については、セクション "分散キャッシュ・クラスタの定義ファイル" に図示しています。

- 基本的なシャード・クラスター ミラーリングされた 3 つのデータ・ノード、アービター・ノード、およびロード・バランサ。この導入については、セクション "シャード・クラスタ定義ファイル" に図示しています。

導入プロセスの手順のほとんどは、どちらの構成でも同じです。主な違いは定義ファイルにあります。詳しい内容は、"導入の定義" を参照してください。プロビジョニング・フェーズ中の出力例 ("icm provision コマンド") は、分散キャッシュ・クラスタのものです。導入フェーズ中の出力例 ("サービスの導入と管理") は、シャード・クラスタのものです。

3.2 ICM の起動

ICM は、Docker イメージとして提供されています。例えば Terraform、Docker クライアント、構成ファイルのテンプレートなど、プロビジョニング、導入、および管理のタスクを実行するために ICM が必要とするすべてが ICM コンテナに含まれています。このため、ICM を起動する Linux、macOS、または Microsoft Windows システムの要件は、Docker がインストールされていることのみです。

- ICM イメージのダウンロード
- ICM コンテナの実行
- ICM コンテナのアップグレード

重要　ICM は、Docker Enterprise Edition および Community Edition バージョン 18.09 以降でサポートされています。実稼働環境でサポートされているのは Enterprise Edition のみです。

注釈　異なるユーザが導入プロセスの異なるフェーズを実行できるようにする場合など、複数の ICM コンテナを使用して 1 つの導入環境を管理できます。詳細は、付録 “ICM 導入環境の共有” を参照してください。

3.2.1 ICM イメージのダウンロード

ICM を使用するには、作業中のシステムに ICM イメージをダウンロードする必要があります。これには、ダウンロード元のレジストリと、アクセスに必要な資格情報を識別することが必要になります。同様に、ICM で InterSystems IRIS およびその他のインターシステムズのコンポーネントを導入するには、関係するイメージのこの情報が必要です。ICM がイメージをダウンロードするレジストリは、使用するクラウド・プロバイダからアクセス可能である (つまり、ファイアウォールの背後にない) ことが必要です。また、セキュリティを確保するために、ICM はユーザから提供された資格情報を使用して認証する必要があります。

InterSystems Container Registry (ICR) には、ICM イメージや InterSystems IRIS イメージを含め、インターシステムズから入手可能なすべてのイメージのリポジトリが含まれています。入手可能なイメージおよび ICR の使用法の詳細は、"InterSystems Container Registry の使用" を参照してください。さらに、組織のプライベート・イメージ・レジストリにインターシステムズのイメージが既に格納されている場合もあります。その場合は、責任のある管理者から、レジストリの場所と認証に必要な資格情報を入手してください。

ICR または組織のレジストリにログインしたら、docker pull コマンドを使用してイメージをダウンロードできます。以下の例は、ICR からのプルを示しています。

```
$ docker login containers.intersystems.com
Username: pmartinez
Password: **********
$ docker pull containers.intersystems.com/intersystems/icm:latest-em
5c939e3a4d10: Pull complete
c63719cdbe7a: Pull complete
19a861ea6baf: Pull complete
651c9d2d6c4f: Pull complete
$ docker images
REPOSITORY                                      TAG             IMAGE ID      CREATED      SIZE
containers.intersystems.com/intersystems/iris 2022.2.0.221.0 15627fb5cb76  1 minute ago 1.39GB
containers.intersystems.com/intersystems/sam  1.0.0.115       15627fb5cb76  3 days ago   1.33GB
acme/centos                                     7.3.1611        262f7381844c  2 weeks ago  192MB
acme/hello-world                                latest          05a3bd381fc2  2 months ag  1.84kB
```

わかりやすくするために、このドキュメントの手順では、**InterSystems** リポジトリからの、**2022.2.0.221.0** タグの付いたインターシステムズのイメージを使用していると想定しています (例 : **intersystems/icm:latest-em**)。

ICM で使用されるイメージが同じレジストリにあるかどうかに関係なく、DockerImage フィールドでレジストリを含むイメージ ID を指定し、イメージごとに DockerUsername フィールドと DockerPassword フィールドで認証に必要な資格情報を指定する必要があります (“InterSystems Cloud Manager の基本的な要素” の章の ″Docker リポジトリ″ を参照)。

3.2.2 ICM コンテナの実行

Docker がインストールされているシステムでコマンド行から ICM を起動するには、docker run コマンド (実際は 3 つの個別の Docker コマンドの組み合わせ) を使用して、以下の操作を行います。

- ICM イメージがまだローカルにない場合、リポジトリからダウンロードします。存在する場合、これは必要に応じて更新されます (docker pull コマンドを使用して、別途、この手順を行うこともできます)。
- ICM イメージからコンテナを作成します (docker create コマンド)。
- ICM コンテナを起動します (docker start コマンド)。

例 :

```
docker run --name icm --init -d -it --cap-add SYS_TIME intersystems/icm:latest-em
```

-i オプションを指定するとコマンドがインタラクティブになり、-t オプションを指定すると疑似 TTY が開いて、コンテナにコマンド行からアクセスできます。ここからは、疑似 TTY コマンド行で ICM コマンドを呼び出して ICM を操作できます。--cap-add SYS_TIME オプションを指定すると、コンテナがホスト・システム上のクロックを操作できます。これにより、クラウド・サービス・プロバイダが API コマンドを拒否する原因になるクロック・スキューが回避されます。

ICM コンテナには /Samples ディレクトリが含まれており、ICM でプロビジョニング、構成、および導入を行うために必要な要素のサンプルが用意されています。/Samples ディレクトリを使用すれば、ICM をすぐに使用したプロビジョニングと導入が容易になります。最終的に、これらの要素と InterSystems IRIS ライセンスを格納するために、コンテナ外部の場所を使用できます。また、ICM 起動時にその場所を外部ボリュームとしてマウントすることも (Docker ドキュメントの ″Manage data in Docker″ を参照)、docker cp コマンドを使用して ICM コンテナにファイルをコピーすることもできます。

もちろん、カスタムのツールとスクリプトを使用して ICM イメージを実行することもできます。これは、例えば、コンテナ内でこのような外部の場所を使用できるようにしたり、また構成ファイルと state ディレクトリ (プロビジョニングしたインフラストラクチャとサービスを削除するために必要) をコンテナ外部の永続ストレージに保存したりするためにも役立つ場合が

あります。例えば以下のように、スクリプトによって現在の作業ディレクトリを変数に取り込み、ICM コンテナの実行時にストレージ・ボリュームとしてそのディレクトリをマウントすることによって、永続ストレージへの保存を行うことができます。

```
#!/bin/bash
clear

# extract the basename of the full pwd path
MOUNT=$(basename $(pwd))
docker run --name icm -d -it --volume $PWD:$MOUNT --cap-add SYS_TIME intersystems/icm:latest-em
printf "\nExited icm container\n"
printf "\nRemoving icm container...\nContainer removed:  "
docker rm icm
```

ICM コンテナ (またはその他のもの) を実行する際に、複数の外部ストレージ・ボリュームをマウントできます。InterSystems IRIS コンテナを導入する際、ICM はいくつかのストレージ・ボリュームのフォーマット、パーティション分割、およびマウントを自動的に行います。詳細は、“ICM リファレンス” の章にある ″ICM によってマウントされるストレージ・ボリューム″ を参照してください。

注釈　Windows ホストでは、Docker の [Settings...] メニューの [Shared Drives] オプションを使用して、ボリュームとしてマウントするディレクトリが配置されているローカル・ドライブを有効にする必要があります。Docker for Windows のその他の要件および一般的な情報については、InterSystems Developer Community の ″Using InterSystems IRIS Containers with Docker for Windows″ を参照してください。

重要　ICM の操作中にエラーが発生した場合、ICM はログ・ファイルを示すメッセージを表示します。このログ・ファイルで、エラーに関する情報を確認できます。ICM の導入を開始する前に、″ログ・ファイルとその他の ICM ファイル″ で説明されている、ログ・ファイルとその場所をよく理解しておいてください。

3.2.3 ICM コンテナのアップグレード

分散管理モードでは、異なるシステム上の異なるユーザが ICM を使用して同じ導入環境を管理でき、ICM コンテナが管理している導入環境の必要な状態ファイルを保持しながら (″状態ディレクトリと状態ファイル″ を参照)、ICM コンテナをアップグレードする手段が提供されます。導入環境を管理している ICM コンテナをアップグレードする際にはこの方法が推奨されるので、分散管理を使用するかどうかに関係なく、このオプションを使用できるように、ICM を使用するたびに分散管理モードを構成することをお勧めします。サービスの検出モードで ICM をアップグレードする方法については、″分散管理モードを使用した ICM のアップグレード″ を参照してください。

3.3 セキュリティ関連ファイルの入手

ICM は、インフラストラクチャをプロビジョニングする対象のクラウド・プロバイダ、プロビジョニングされた各ノードのオペレーティング・システム、およびコンテナの導入後は Docker および複数の InterSystems IRIS サービスとの間で、セキュリティ保護された通信を行います。導入を定義する前に、セキュリティ保護された通信を有効にするために必要な資格情報とその他のファイルを入手する必要があります。

3.3.1 クラウド・プロバイダの資格情報

パブリック・クラウド・プラットフォームのいずれかと組み合わせて ICM を使用するには、アカウントを作成して管理用の資格情報をダウンロードする必要があります。このためには、クラウド・プロバイダが提供する指示に従ってください。また、アカウントを作成した後で資格情報をダウンロードする方法については、“ICM リファレンス” の章の ″プロバイダ固有のパラメータ″ のセクションに説明があります。ICM 構成ファイルで、プロバイダに固有のパラメータを使用してこれらの資格情報の場所を指定します。AWS の場合は、Credentials パラメータを使用します。

vSphere プライベート・クラウドと組み合わせて ICM を使用する場合は、必要な特権を持つ既存のアカウントを使用することも、新規アカウントを作成することもできます。これらの情報は、Username フィールドと Password フィールドを使用して指定します。

3.3.2 SSH 鍵と TLS 鍵

ICM は、プロビジョニングされたノードのオペレーティング・システムへのセキュリティ保護されたアクセスを提供するために SSH を使用し、Docker、インターシステムズの Web ゲートウェイ、および JDBC への接続と、InterSystems IRIS のミラー、分散キャッシュ・クラスタ、およびシャード・クラスタ内のノード間のセキュリティ保護された接続を確立するために TLS を使用します。このセキュリティ保護された通信を有効にするために必要なファイルの場所は、以下のようないくつかの ICM パラメータを使用して指定します。

- SSHPublicKey

  プロビジョニングされたホスト・ノードへのセキュリティ保護された接続を有効にするために使用される SSH 公開/秘密鍵ペアの公開鍵。AWS の場合は SSH2 形式、その他のプロバイダの場合は OpenSSH 形式です。

- SSHPrivateKey

  SSH 公開/秘密鍵ペアの秘密鍵。

- TLSKeyDir

  Docker、インターシステムズの Web ゲートウェイ、JDBC、およびミラーリングされた InterSystems IRIS データベースへのセキュリティ保護された接続を確立するために使用される TLS 鍵を含むディレクトリ。

これらのファイルを作成して ICM で使用でき、またこれらのファイルを検討してどれが必要か確認できます。このためには、ICM コンテナ内のディレクトリ /ICM/bin に用意されている 2 つのスクリプトを使用します。keygenSSH.sh スクリプトは、必要な SSH ファイルを作成して、ICM コンテナ内のディレクトリ /Samples/ssh に配置します。keygenTLS.sh スクリプトは必要な TLS ファイルを作成し、/Samples/tls に配置します。導入を定義する際にこれらの場所を指定でき、これらのディレクトリの内容に基づいてユーザが独自にファイルを取得することもできます。

keygen* スクリプトによって生成される、ICM で必要なセキュリティ・ファイルの詳細は、“ICM リファレンス” の章の ″ICM セキュリティ″ および ″セキュリティ関連のパラメータ″ を参照してください。

重要

これらのスクリプトによって生成される鍵、およびクラウド・プロバイダの資格情報により、これらが使用される ICM 導入環境へのフルアクセスが可能になるので、これらの情報は完全にセキュリティ保護する必要があります。

keygen* スクリプトによる鍵は、初期テスト導入でユーザが便利に使用できるように生成されます。(一部は、インターシステムズ社に固有の文字列を含んでいます。)プロダクション環境では、必要な鍵は企業のセキュリティ・ポリシーに準拠して生成または取得する必要があります。

3.4 導入の定義

必要なパラメータを ICM に指定するには、目的と状況に応じていくつかのフィールドの値を選択した後、導入に使用する既定値ファイルと定義ファイルにこれらの値を取り込む必要があります。ICM コンテナ内の /Samples ディレクトリ・ツリー (例えば /Samples/AWS) で提供されているテンプレート defaults.json ファイルと definitions.json ファイルから始めることができます。

″構成ファイル、状態ファイル、およびログ・ファイル″ に記載されているように、defaults.json は多くの場合、特定のカテゴリに属する複数の導入環境 (同じプラットフォーム上でプロビジョニングされるものなど) の共有設定を提供するために使用されるのに対し、別個の definitions.json ファイルでは、それぞれの導入環境についてプロビジョニングして構成する必要があるノード・タイプを定義します。例えば、ここに示す別個の定義ファイルでは、この章の冒頭で説明した 2 つ

のターゲット導入環境を定義します。分散キャッシュ・クラスタには、ミラーリングされたデータ・サーバとしての DM ノードが 2 つ、アプリケーション・サーバとしての負荷分散 AM ノードが 3 つ、およびアービター (AR) ノードが 1 つ含まれ、シャード・クラスタには、ミラーリングされた 3 つの負荷分散データ・ノードとして構成される DATA ノードが 6 つとアービター・ノードが 1 つ含まれています。同時に、それぞれの導入環境では多数の特性が共通しているため、defaults.json ファイルを共有できます。例えば、これらの導入環境はどちらも AWS 上にあり、同じ資格情報を使用します。また、同じリージョンおよびアベイラビリティ・ゾーンにプロビジョニングされ、同じ InterSystems IRIS イメージを導入します。

Provider などの一部のフィールドは defaults.json で指定し、Role などの一部のフィールドは definitions.json で指定する必要がありますが、その他のフィールドは、ニーズに応じてどちらでも使用できます。この場合は、例えば、InstanceType フィールドは共有の既定値ファイルと両方の定義ファイルで指定します。DM、AM、DATA、および AR の各ノードはすべて、異なる計算リソースを必要とするので、1 つの defaults.json 設定で既定のインスタンス・タイプを設定するだけでは十分ではありません。

以下のセクションでは、導入する InterSystems IRIS インスタンスの構成をカスタマイズする方法について説明すると共に、共有の既定値ファイルと別個の定義ファイルの両方の内容を検討します。それぞれのフィールド/値のペアは、構成ファイルに指定されるとおりに記載されています。

- 共有の既定値ファイル
- 分散キャッシュ・クラスタの定義ファイル
- シャード・クラスタの定義ファイル
- InterSystems IRIS 構成のカスタマイズ

ICM を使用すると、既存の導入環境の定義ファイルを変更した後、再プロビジョニングや再導入を行ってノードを追加または削除したり、既存のノードを変更したりすることができます。詳細は、"インフラストラクチャの再プロビジョニング" および "サービスの再導入" を参照してください。

重要　フィールド名と値の両方で大文字と小文字が区別されます。例えば、クラウド・プロバイダとして AWS を選択するには、既定値ファイルで、“provider”:”AWS” や “Provider”:”aws” などではなく “Provider”:”AWS” と指定する必要があります。

注釈　ここに含まれているフィールドは、適用される可能性があるフィールドのサブセットを表します。必須のフィールドとオプションのフィールド (一般的なものとプロバイダ固有のものの両方を含む) すべての包括的なリストについては、"ICM の構成パラメータ" を参照してください。

3.4.1 共有の既定値ファイル

このセクションのテーブルに示すフィールド/値のペアは、分散キャッシュ・クラスタ導入とシャード・クラスタ導入の両方に使用できる defaults.json ファイルの内容を表しています。このセクションの冒頭で説明したように、以下に示す /Samples/AWS/default.json ファイルにいくつかの変更を加えることで、このファイルを作成できます。

```
{
    "Provider": "AWS",
    "Label": "Sample",
    "Tag": "TEST",
    "DataVolumeSize": "10",
    "SSHUser": "ubuntu",
    "SSHPublicKey": "/Samples/ssh/insecure-ssh2.pub",
    "SSHPrivateKey": "/Samples/ssh/insecure",
    "DockerRegistry": "https://containers.intersystems.com",
    "DockerImage": "containers.intersytems.com/intersystems/iris:some-tag",
    "DockerUsername": "xxxxxxxxxxxx",
    "DockerPassword": "xxxxxxxxxxxx",
    "TLSKeyDir": "/Samples/tls/",
    "LicenseDir": "/Samples/license/",
    "Region": "us-east-1",
    "Zone": "us-east-1a",
    "AMI": "ami-07267eded9a267f32",
    "DockerVersion": "5:20.10.17~3-0~ubuntu-jammy",
    "InstanceType": "m5.large",
    "Credentials": "/Samples/AWS/sample.credentials",
```

```
    "ISCPassword": "",
    "Mirror": "false",
    "UserCPF": "/Samples/cpf/iris.cpf"
}
```

テーブル内のフィールドの順序は、このサンプル既定値ファイルの順序と一致しています。

別のプロバイダの既定値ファイルでは、プロバイダ固有の異なる値が指定されるフィールドもあれば、プロバイダ固有の異なるフィールドに置き換えられるフィールドもあります。例えば、Tencent の既定値ファイルでは、InstanceType の値は S2.MEDIUM4 です。これは Tencent 固有のインスタンス・タイプであり、AWS では無効です。また、AWS の AMI フィールドは、Tencent の同等のフィールドである ImageId に置き換えられます。/Samples ディレクトリ・ツリー内のさまざまな defaults.json ファイルを調べ、“ICM リファレンス” の章にある ″一般パラメータ″ と ″プロバイダ固有のパラメータ″ のテーブルを参照することによって、これらの違いを確認できます。

注釈　このサンプル既定値ファイルでセキュリティ・ファイルを指定するフィールドに示されているパス名は、AWS 資格情報を /Samples/AWS ディレクトリに配置していること、また、″セキュリティ関連ファイルの入手″ の説明に従って、keygen*.sh スクリプトを使用して鍵を生成していることを前提としています。独自の鍵を生成または取得した場合、これらのパスは、ICM コンテナの実行時にマウントされる外部ストレージ・ボリュームの内部パスに置き換えることもできます (″ICM の起動″ を参照)。これらのファイルに関する追加情報は、“ICM リファレンス” の章にある ″ICM セキュリティ″ および ″セキュリティ関連のパラメータ″ を参照してください。

共通の特性	/Samples/AWS/defaults.json	カスタマイズの例	カスタマイズの説明
インフラストラクチャをプロビジョニングするプラットフォーム (この場合は Amazon Web Services)。“基本的な ICM の要素” の章にある ″プロビジョニング・プラットフォーム″ を参照してください。	″Provider″: ″AWS″,	N/A	値を GCP、Azure、Tencent、vSphere、または PreExisting に変更した場合は、ここに示されているものとは異なるフィールドと値が必要になります。
プロビジョニングされるノードの命名パターンは Label-Role-Tag-NNNN です。Role は、定義ファイル内の Role フィールドの値です (例 : ANDY-DATA-TEST-0001)。ノード名が所有権と目的を示すように、これらを変更します。	″Label″: ″Sample″, ″Tag″: ″TEST″,	″Label″: ″ANDY″, ″Tag″: ″TEST″,	所有者を示すように更新します。

共通の特性	/Samples/AWS/defaults.json	カスタマイズの例	カスタマイズの説明
InterSystems IRIS コンテナ用に作成する永続データ・ボリュームのサイズ (GB 単位)。“ICM リファレンス” の章にある "ICM によってマウントされるストレージ・ボリューム" を参照してください。定義ファイルで特定のノード・タイプについてオーバーライドできます。	"DataVolumeSize": "10",	"DataVolumeSize": "250",	既定値ファイルを使用するすべての導入環境がシャード・クラスタ (DATA) ノードのみで構成される場合は、データ・ボリュームの既定のサイズを大きくすることをお勧めします。
プロビジョニングされたノードへの SSH アクセスのために ICM によって使用される、sudo アクセス権を持つ非 root アカウント。AWS の場合、必要な値は AMI によって異なりますが、Ubuntu AMI の場合は通常 ubuntu になります。“ICM リファレンス” の章にある "セキュリティ関連のパラメータ" を参照してください。	"SSHUser": "ubuntu",	N/A	値を GCP、Azure、Tencent、vSphere、または PreExisting に変更した場合は、ここに示されているものとは異なるフィールドと値が必要になります。
必要なセキュリティ・キー・ファイルの場所。"セキュリティ関連ファイルの入手" および "セキュリティ関連のパラメータ" を参照してください。プロバイダが AWS なので、/Samples/ssh/ 内の SSH2 形式の公開鍵が指定されています。	"SSHPublicKey": "/Samples/ssh/secure-ssh2.pub", "SSHPrivateKey": "/Samples/ssh/secure-ssh2", "TLSKeyDir": "/Samples/tls/",	"SSHPublicKey": "/mydir/keys/mykey.pub", "SSHPrivateKey": "/mydir/keys/mykey.ppk", "TLSKeyDir": "/mydir/keys/tls/",	マウントされた外部ボリュームで鍵をステージングする場合は、これを反映するようにパスを更新します。
プロビジョニングされるノードにインストールされる Docker バージョン。通常、既定値のままにできます。	"DockerVersion": "5:20.10.5~3-0~ubuntu-bionic",	"DockerVersion": "18.06.1~ce~3-0~ubuntu",	各 /Samples/.../defaults.json のバージョンは通常、プラットフォームに適合しています。ただし、組織で別のバージョンの Docker を使用している場合は、代わりに、そのバージョンをクラウド・ノードにインストールできます。

共通の特性	/Samples/AWS/defaults.json	カスタマイズの例	カスタマイズの説明
プロビジョニングされるノードに導入する Docker イメージ。“InterSystems Cloud Manager の基本的な要素” の章の “Docker リポジトリ”、“ICM リファレンス” の章の “icm run コマンド” と “一般パラメータ” を参照。このフィールドを definitions.json のノード定義に含めて、既定値ファイルの値をオーバーライドすることもできます。(“分散キャッシュ・クラスタの定義ファイル” を参照)。	“DockerImage”: “intersystems/iris:stable”,	“DockerImage”: “acme/iris:latest-em”	InterSystems IRIS イメージを組織のレジストリにプッシュした場合は、イメージの指定を更新します。 注意 : 標準プラットフォームの InterSystems IRIS イメージには、**iris** という名前が付けられます。ARM プラットフォームの場合は **iris-arm64** という名前が付けられます。
前のフィールドで指定したイメージを格納する Docker レジストリにログインするための資格情報。“ICM イメージのダウンロード” を参照。	“DockerUsername”: “xxxxxxxxxxx”, “DockerPassword”: “xxxxxxxxxxx”,	“DockerUsername”: “AndyB”, “DockerPassword”: “password”,	指定したレジストリに独自の Docker 資格情報を使用するように更新します。
ICM コンテナ内でステージングされ、定義ファイルの LicenseKey フィールドによって個々に指定される InterSystems IRIS ライセンス・キーの場所。“ICM リファレンス” の章にある “ICM の InterSystems IRIS ライセンス” を参照してください。	“LicenseDir”: “/Samples/Licenses”,	“LicenseDir”: “/mydir/licenses”,	マウントされた外部ボリュームでライセンスをステージングする場合は、これを反映するようにパスを更新します。
インフラストラクチャがプロビジョニングされる、プロバイダの計算リソースの地理的地域。“一般パラメータ” を参照してください。	“Region”: “us-west-1”,	“Region”: “us-east-2”,	リージョンとアベイラビリティ・ゾーンの別の有効な組み合わせでプロビジョニングする場合は、これを反映するように値を更新します。
プロビジョニングされるノードを配置する、指定したリージョン内のアベイラビリティ・ゾーン。“一般パラメータ” を参照してください。	“Zone”: “us-west-1c”,	“Zone”: “us-east-2a”,	

共通の特性	/Samples/AWS/defaults.json	カスタマイズの例	カスタマイズの説明
プロビジョニングされるノードのプラットフォームおよび OS のテンプレートとして使用する AMI。“ICM リファレンス” の章にある ″Amazon Web Services (AWS) パラメータ″ を参照してください。	″AMI″: ″ami-0121ef35996ede438″,	″AMI″: ″ami-e24b7d9d″,	AMI とインスタンス・タイプの別の有効な組み合わせからプロビジョニングする場合は、これを反映するように値を更新します。
プロビジョニングされるノードの計算リソースのテンプレートとして使用するインスタンス・タイプ。″Amazon Web Services (AWS) パラメータ″ を参照してください。	″InstanceType″: ″m4.large″,	″InstanceType″: ″m5ad.large″,	
AWS アカウントの資格情報。″Amazon Web Services (AWS) パラメータ″ を参照してください。	″Credentials″: ″/Samples/AWS/sample.credentials″,	“Credentials”: “/mydir/aws-credentials”,	マウントされた外部ボリュームで資格情報をステージングする場合は、これを反映するようにパスを更新します。
導入される InterSystems IRIS インスタンスのパスワード。推奨されるアプローチは、導入用コマンド行で指定して (″サービスの導入と管理″ を参照)、構成ファイルにパスワードが表示されないようにすることです。	″ISCPassword″: ″″,	(削除)	削除して、icm run コマンドの -password オプションを使用してパスワードを指定するようにします。
偶数個で定義された特定のノード・タイプ (DM および DATA を含む) をミラーとして導入するかどうか (″ミラーリングの規則″ を参照)。	″Mirror″: ″true″	N/A	両方の導入環境がミラーリングされます。

共通の特性	/Samples/AWS/defaults.json	カスタマイズの例	カスタマイズの説明
導入される InterSystems IRIS インスタンスの初期 CPF 設定をオーバーライドするために使用される構成マージ・ファイル (“ICM リファレンス” の章にある ″カスタマイズされた InterSystems IRIS 構成を使用した導入″ を参照)。	″UserCPF″: ″/Samples/cpf/iris.cpf″		構成マージ機能および CPF 設定に精通している場合を除き、削除します (″構成マージを使用した InterSystems IRIS の自動構成″ を参照)。

重要　ICM を起動したイメージと **DockerImage** フィールドを使用して指定した InterSystems IRIS イメージのメジャー・バージョンが一致している必要があります。例えば、2022.1 バージョンの ICM を使用して 2022.2 バージョンの InterSystems IRIS を導入することはできません。インターシステムズのコンテナをアップグレードする前に ICM をアップグレードする方法については、付録 “分散管理モードでの導入環境の共有” の ″分散管理モードを使用した ICM のアップグレード″ を参照してください。

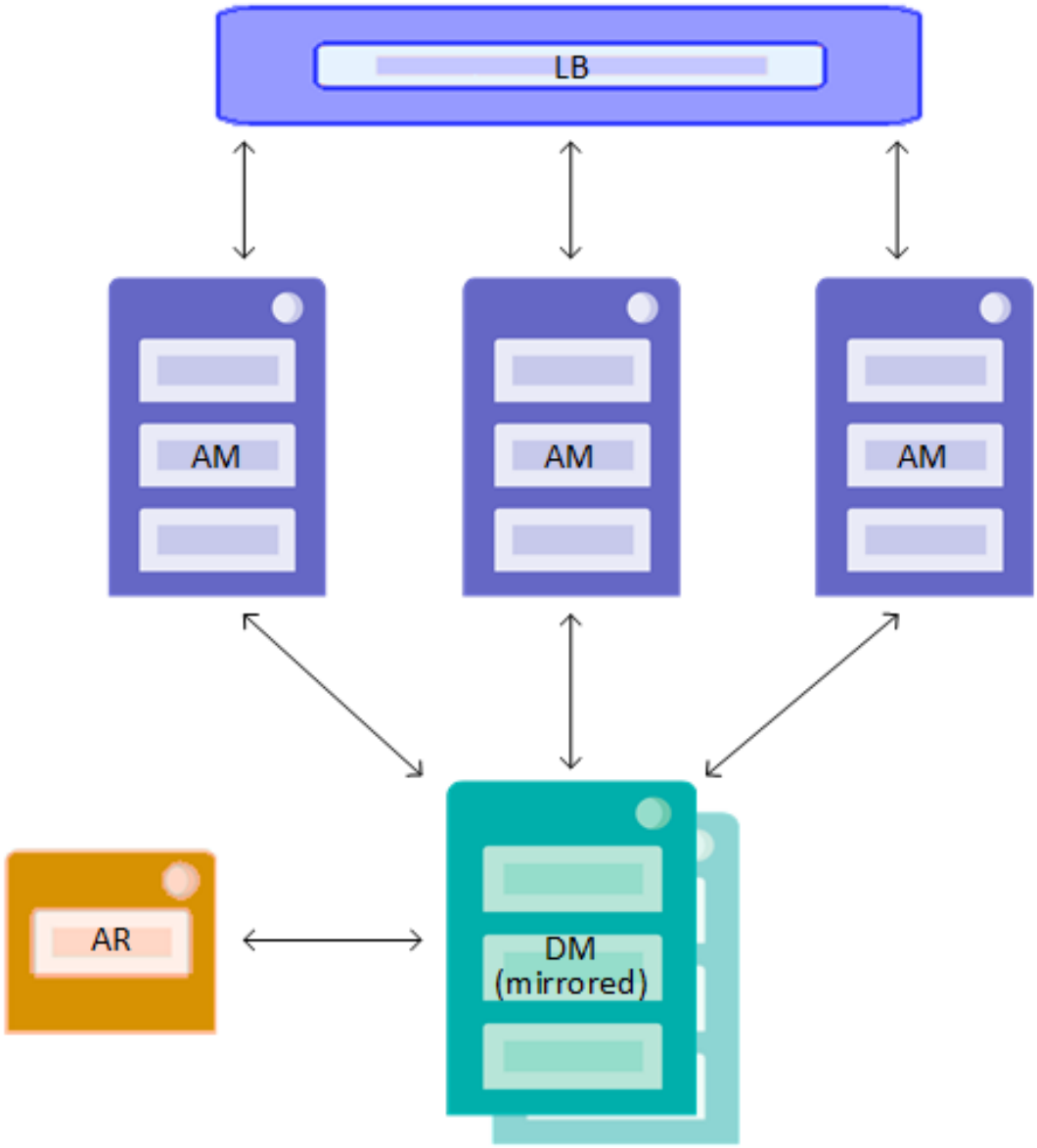
3.4.2 分散キャッシュ・クラスタの定義ファイル

分散キャッシュ・クラスタの **definitions.json** ファイルでは、以下のノードを定義する必要があります。

- ミラーとして構成される 2 つのデータ・サーバ (ロール DM)
- 3 つのアプリケーション・サーバ (ロール AM)
- アプリケーション・サーバ用のロード・バランサ
- データ・サーバ・ミラー用のアービター・ノード

この構成の図を以下に示します。

図 3-1: ICM によって導入される分散キャッシュ・クラスタ



以下の表に、この構成に必要なフィールド/値のペアをリストします。

重要　スタンドアロン InterSystems IRIS インスタンスは、ミラーリングの有無によらず (すなわち、1 つの DM ノードでもミラーを構成する 2 つの DM ノードでも)、分散キャッシュ・クラスタ (DM ノードまたはミラーリングされた DM ノードと AM ノード) と同様に、標準ライセンスで導入できます。すべてのシャード・クラスタ構成では (ノード・レベルまたはネームスペース・レベル、およびミラーリングなしまたはあり)、InterSystems IRIS コンテナが導入されているすべてのノードに、シャーディング対応 InterSystems IRIS ライセンスが必要です。例えば、標準ライセンスを持つミラーリングなし、またはありのスタンドアロン・インスタンスに、DS ノードを追加した場合、すべてのノードに使用されているライセンスをシャーディング・ライセンスにアップグレードする必要があります。

定義	フィールド: 値
ミラーとして構成される (共有の既定値ファイルで "Mirror": "true" と指定されているため)、標準 InterSystems IRIS ライセンスを使用する 2 つのデータ・サーバ (DM)。 インスタンス・タイプ、OS ボリューム・サイズ、データ・ボリューム・サイズでは、データ・サーバのリソース要件に合わせて既定値ファイルの設定をオーバーライドします。	"Role": "DM", "Count": "2", "LicenseKey": "ubuntu-standard-iris.key," "InstanceType": "m4.xlarge", "OSVolumeSize": "32", "DataVolumeSize": "150",
標準 InterSystems IRIS ライセンスを使用する 3 つのアプリケーション・サーバ (AM)。 ノード名の番号は、DM ノードの 0001 および 0002 に続いて 0003 から始まります。 アプリケーション・サーバ用のロード・バランサが自動的にプロビジョニングされます。	"Role": "AM", "Count": "3", "LicenseKey": "ubuntu-standard-iris.key", "StartCount": "3", "LoadBalancer": "true",
データ・サーバ・ミラー用の 1 つのアービター (AR)。ライセンスは不要です。アービター・イメージを使用して、既定値ファイルで指定された InterSystems IRIS イメージをオーバーライドします。 ノードの番号は 0006 です。 アービターでは限られたリソースのみが必要なので、インスタンス・タイプで既定値ファイルをオーバーライドします。	"Role": "AR", "Count": "1", "DockerImage": "intersystems/arbiter:latest-em", "StartCount": "6", "InstanceType": "t2.small",

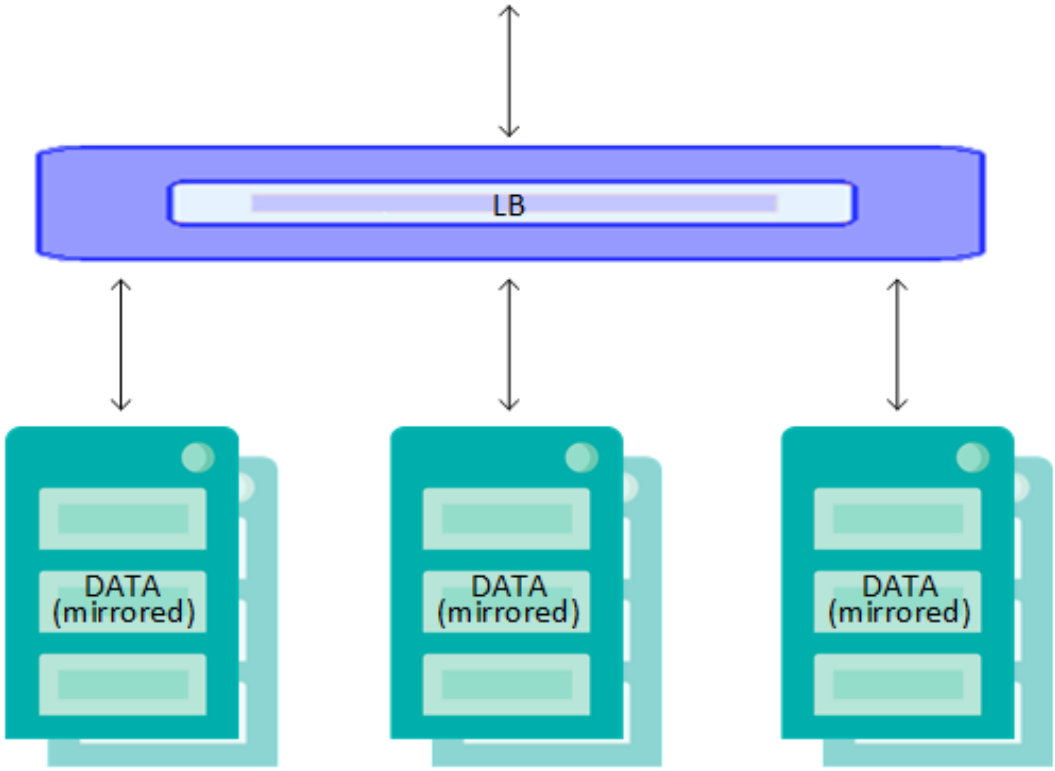
上のテーブルの設定を組み込んだ definitions.json ファイルは以下のようになります。

```
[
    {
        "Role": "DM",
        "Count": "2",
        "LicenseKey": "ubuntu-standard-iris.key",
        "InstanceType": "m4.xlarge",
        "OSVolumeSize": "32",
        "DataVolumeSize": "150"
    },
    {
        "Role": "AM",
        "Count": "3",
        "LicenseKey": "ubuntu-standard-iris.key",
        "StartCount": "3",
        "LoadBalancer": "true"
    },
    {
        "Role": "AR",
        "Count": "1",
        "DockerImage": "intersystems/arbiter:latest-em",
        "StartCount": "6",
        "InstanceType": "t2.small"
    }
]
```

3.4.3 シャード・クラスタの定義ファイル

シャード・クラスタ構成の definitions.json ファイルでは、ミラーリングされた 3 つの負荷分散 DATA ノードを定義する必要があります。これを以下に示します。

図 3-2: ICM によって導入されるシャード・クラスタ



以下の表に、この構成に必要なフィールド/値のペアをリストします。

定義	フィールド: 値
• 3 つのミラーとして構成される (共有の既定値ファイルで "Mirror": "true" と指定されているため)、シャーディング対応 InterSystems IRIS ライセンスを使用する 6 つのデータ・ノード (DATA)。 • インスタンス・タイプとデータ・ボリューム・サイズでは、データ・ノードのリソース要件に合わせて既定値ファイルの設定をオーバーライドします。 • データ・ノード用のロード・バランサが自動的にプロビジョニングされます。	"Role": "DATA" "Count": "6" "LicenseKey": "ubuntu-sharding-iris.key" "InstanceType": "m4.4xlarge" "DataVolumeSize": "250" "LoadBalancer": "true"
• データ・サーバ・ミラー用の 1 つのアービター (AR)。ライセンスは不要です。アービター・イメージを使用して、既定値ファイルで指定された InterSystems IRIS イメージをオーバーライドします。 • ノードの番号は 0007 です。 • アービターでは限られたリソースのみが必要なので、インスタンス・タイプで既定値ファイルをオーバーライドします。	"Role": "AR" "Count": "1" "DockerImage": "intersystems/arbiter:latest-em" "StartCount": "7" "InstanceType": "t2.small"

上のテーブルの設定を組み込んだ definitions.json ファイルは以下のようになります。

```
[
    {
        "Role": "DATA",
        "Count": "6",
        "LicenseKey": "sharding-iris.key”,
        "InstanceType": "m4.xlarge",
        "DataVolumeSize": "250",
        "LoadBalancer": "true"
    },
    {
        "Role": "AR",
        "Count": "1",
        "DockerImage": "intersystems/arbiter:latest-em",
        "StartCount": "7",
        "InstanceType": "t2.small"
    }
]
```

注釈　ノード・レベルのシャーディング・アーキテクチャをサポートするために、リリース 2019.3 で DATA ノード・タイプと COMPUTE ノード・タイプが ICM に追加されました。このドキュメントの以前のバージョンでは、より大規模な別のノード・タイプのセットを含むネームスペース・レベルのシャーディング・アーキテクチャについて説明していました。ネームスペース・レベルのアーキテクチャは、ノード・レベルのアーキテクチャの透過的な基盤として残っており、ノード・レベルのアーキテクチャと完全な互換性があるため、このアーキテクチャを導入するために使用されるノード・タイプは ICM で引き続き使用可能です。使用可能なすべてのノード・タイプについては、″ICM ノード・タイプ″ を参照してください。

データベース・キャッシュ・サイズやデータ・ボリューム・サイズの要件など、シャード・クラスタの具体的な導入の詳細は、″スケーラビリティ・ガイド″ の “シャーディングによるデータ量に応じた水平方向の拡張” の章にある ″シャード・クラスタの導入″ を参照してください。

ベスト・プラクティスとして、クラスタ内のすべてのデータ・ノード間でアプリケーション接続を負荷分散することをお勧めします。

3.4.4 InterSystems IRIS 構成のカスタマイズ

ICM によって導入されたコンテナ内で実行されるものも含め、InterSystems IRIS インスタンスはすべて、事前に指定された一連の構成設定を使用してインストールされます。これらの設定は、構成パラメータ・ファイル (CPF) に記録されています。既定値ファイルの UserCPF フィールドを使用して (″共有の既定値ファイル″ の /Samples/AWS/defaults.json ファイルを参照)、構成マージ・ファイルを指定できます。これにより、導入するすべての InterSystems IRIS インスタンスについて 1 つ以上の構成設定をオーバーライドできます。また、分散キャッシュ・クラスタ内の DM ノードや AM ノード、シャード・クラスタ内の DATA ノードや COMPUTE ノードなど、それぞれのノード・タイプについて異なる設定を定義ファイルでオーバーライドすることもできます。例えば、″スケーラビリティ・ガイド″ の “シャーディングによるデータ量に応じた水平方向の拡張” の章にある ″InterSystems IRIS シャード・クラスタの計画″ に記載されているように、シャード・クラスタ内のデータ・ノードのデータベース・キャッシュのサイズを調整することができます。そのためには、DATA の定義のみについて [config]/globals CPF 設定の値をオーバーライドします。マージ・ファイルを使用して初期 CPF 設定をオーバーライドする方法については、“ICM リファレンス” の章にある ″カスタマイズされた InterSystems IRIS 構成を使用した導入″ を参照してください。

簡単な構成マージ・ファイルが ICM コンテナの /Samples/cpf ディレクトリに用意されており、すべての /Samples プロバイダ・サブディレクトリ内のサンプル既定値ファイルに、このファイルを指す UserCPF フィールドが含まれています。導入される InterSystems IRIS インスタンスの既定の CPF にその内容をマージすることが確実な場合を除き、既定値ファイルから UserCPF を削除してください。

InterSystems IRIS の構成設定、それらの影響、およびインストールされる既定値については、″インストール・ガイド″、″システム管理ガイド″、および ″構成パラメータ・ファイル・リファレンス″ を参照してください。

3.5 インフラストラクチャのプロビジョニング

ICM は、HashiCorp Terraform ツールを使用してクラウド・インフラストラクチャをプロビジョニングします。

- icm provision コマンド
- インフラストラクチャの再プロビジョニング
- インフラストラクチャ管理コマンド

注釈　ICM では、既存のクラウド・インフラストラクチャ、仮想インフラストラクチャ、または物理インフラストラクチャにコンテナを導入できます。詳細は、"既存のクラスタへの導入" を参照してください。

3.5.1 icm provision コマンド

icm provision コマンドは、definitions.json ファイルと defaults.json ファイルに指定されたフィールド値、および該当する場合は指定されていないパラメータの既定値を使用して、ホスト・ノードを割り振り、構成します。現在の作業ディレクトリの入力ファイルが使用され、プロビジョニング時に、同じディレクトリに state ディレクトリ、instances.json ファイル、およびログ・ファイルが作成されます (これらのファイルの詳細は、" 基本的な ICM の要素 " の章にある "構成ファイル、状態ファイル、およびログ・ファイル" を参照してください)。

state ディレクトリと生成される instances.json ファイル (後続の再プロビジョニング、導入、および管理コマンドへの入力として使用される) は導入環境に固有なので、通常は、導入環境ごとにディレクトリを設定して作業するのが最も簡単で効果的なアプローチです。例えば、ここで説明するシナリオ (2 つの異なる導入環境が既定値ファイルを共有する) における最も簡単なアプローチは、"共有の既定値ファイル" に示されているように、/Samples/AWS などのディレクトリで 1 つの導入環境を設定した後、そのディレクトリ全体を (/Samples/AWS2 などに) コピーし、1 つ目の definitions.json ファイルを 2 つ目の定義ファイルに置き換えることです。

プロビジョニング操作の進行中、ICM は計画フェーズ (目的のインフラストラクチャを検証し、状態ファイルを生成する Terraform のフェーズ)、および適用フェーズ (クラウド・プロバイダにアクセスし、マシンの割り振りを実行し、状態ファイルを更新する Terraform のフェーズ) に関する状態メッセージを出します。ICM は複数のスレッドで Terraform を実行するので、マシンがプロビジョニングされる順序と、これらのマシンに追加のアクションが適用される順序は不定です。この例は、以下のサンプル出力に示されています。

また、完了時に、ICM はプロビジョニングされたホスト・ノードと関連コンポーネントの要約を示し、後でインフラストラクチャをプロビジョニング解除 (削除) するために使用できるコマンド行を出力します。

以下の例は、"導入の定義" に記載されている分散キャッシュ・クラスタのプロビジョニングの出力から抜粋したものです。

```
$ icm provision
Starting init of ANDY-TEST...
...completed init of ANDY-TEST
Starting plan of ANDY-DM-TEST...
...
Starting refresh of ANDY-TEST...

...
Starting apply of ANDY-DM-TEST...
...
Copying files to ANDY-DM-TEST-0002...
...
Configuring ANDY-AM-TEST-0003...
...
Mounting volumes on ANDY-AM-TEST-0004...
...
Installing Docker on ANDY-AM-TEST-0003...
...
Installing Weave Net on ANDY-DM-TEST-0001...
...
Collecting Weave info for ANDY-AR-TEST-0006...
...
...collected Weave info for ANDY-AM-TEST-0005
```

```
...installed Weave Net on ANDY-AM-TEST-0004

Machine            IP Address     DNS Name                                                  Region    Zone
-------            ----------     --------                                                  ------    ----
ANDY-DM-TEST-0001+ 00.53.183.209  ec2-00-53-183-209.us-west-1.compute.amazonaws.com  us-west-1 c
ANDY-DM-TEST-0002- 00.53.183.185  ec2-00-53-183-185.us-west-1.compute.amazonaws.com  us-west-1 c
ANDY-AM-TEST-0003  00.56.59.42    ec2-00-56-59-42.us-west-1.compute.amazonaws.com    us-west-1 c
ANDY-AM-TEST-0005  00.67.1.11     ec2-00-67-1-11.us-west-1.compute.amazonaws.com     us-west-1 c
ANDY-AM-TEST-0003  00.193.117.217 ec2-00-193-117-217.us-west-1.compute.amazonaws.com us-west-1 c
ANDY-LB-TEST-0002  (virtual AM)   ANDY-AM-TEST-1546467861.amazonaws.com              us-west-1 c
ANDY-AR-TEST-0006  00.53.201.194  ec2-00-53-201-194.us-west-1.compute.amazonaws.com  us-west-1 c
To destroy: icm unprovision [-cleanUp] [-force]
```

重要　　適切な時期にパブリック・クラウドのホスト・ノードをプロビジョニング解除することで、不要な出費を防ぐことができます。

クラウド・プロバイダとのやり取りに長い待ち時間がかかって、プロバイダ側でタイムアウトや内部エラーが発生することがあります。また、構成ファイル内のエラーが原因でプロビジョニングに失敗することもあります。icm provision コマンドは完全に再入可能なので、ICM が指定ノードすべてに対してすべての必須タスクをエラーなしで完了するまで複数回発行できます。詳細は、次の ″インフラストラクチャの再プロビジョニング″ のセクションを参照してください。

3.5.2 インフラストラクチャの再プロビジョニング

プロビジョニング・プロセスの柔軟性と回復力を可能な限り高めるために、icm provision コマンドは完全に再入可能であり、同じ導入環境について複数回発行できます。icm provision コマンドを複数回実行してインフラストラクチャを再プロビジョニングする主な理由としては、以下の 2 つがあります。

- プロビジョニング・エラーの解決

  クラウド・プロバイダとのやり取りに長い待ち時間がかかって、プロバイダ側でタイムアウトや内部エラーが発生することがあります。プロビジョニング中にエラーが発生した場合、ICM が指定ノードすべてに対してすべての必須タスクをエラーなしで完了するまで、コマンドを複数回発行できます。

- プロビジョニングされたインフラストラクチャの変更

  変更が必要な場合、既存のノードの特性の変更、ノードの追加、またはノードの削除を行うことによって、サービスが導入された構成を含め、既にプロビジョニングされたインフラストラクチャをいつでも変更できます。

エラーが発生した後に icm provision コマンドを繰り返す場合に、作業ディレクトリに構成ファイルが含まれていないときは、場所をオーバーライドするオプションを繰り返す必要があります。このファイルはまだ存在しないため、-stateDir オプションを使用して、プロビジョニングを続行する不完全なインフラストラクチャを指定する必要があります。ただし、正常にプロビジョニングされたインフラストラクチャを変更するためにこのコマンドを繰り返すときには、その必要はありません。instances.json ファイルが含まれているディレクトリで作業している限り、自動的にそれを使用して、再プロビジョニングするインフラストラクチャが特定されます これについては、この後のセクションで示します。

3.5.2.1 プロビジョニング・エラーの解決

icm provision コマンドを発行したときに、エラーが発生してプロビジョニングが正常に完了しなかった場合、state ディレクトリは作成されますが、instances.json ファイルは作成されません。icm provision コマンドを再度発行する際に、現在の作業ディレクトリに state サブディレクトリがない場合は、-stateDir オプションを使用してサブディレクトリの場所を指定します。これにより、プロビジョニングが不完全であることが示され、完了した操作と完了していない操作に関する必要な情報が提供されます。例えば、以下に示す問題が発生したとします。

```
$ icm provision
Starting plan of ANDY-DM-TEST...
...completed plan of ANDY-DM-TEST
Starting apply of ANDY-AM-TEST...
Error: Thread exited with value 1
See /Samples/AWS/state/Sample-DS-TEST/terraform.err
```

示されたエラーを確認し、必要に応じて修正してから、同じディレクトリで icm provision を再度実行します。

```
$ icm provision
Starting plan of ANDY-DM-TEST...
...completed plan of ANDY-DM-TEST
Starting apply of ANDY-DM-TEST...
...completed apply of ANDY-DM-TEST
[...]
To destroy: icm unprovision [-cleanUp] [-force]
```

3.5.2.2 プロビジョニングされたインフラストラクチャの変更

icm run コマンドを使用してサービスが正常に導入された後など、プロビジョニングが正常に完了した後にいつでも、definitions.json ファイルを変更し、icm provision コマンドを再度実行することによって、プロビジョニングされたインフラストラクチャや構成を変更できます 導入済みの構成を変更する場合は、"サービスの再導入" の説明に従って、icm run コマンドを再度実行します。

既存のインフラストラクチャまたは導入済みの構成を以下のように変更することができます。

- 1 つ以上のノードの特性を変更するには、定義ファイルのノード定義内の設定を変更します。ノードを垂直方向に拡張するような場合に、この作業を行います。例えば、以下の定義で、DataVolumeSize 設定 ("一般パラメータ" を参照) を変更して、DM ノードのデータ・ボリュームのサイズを大きくすることができます。

  ```
  {
      "Role": "DM",
      "Count": "2",
      "LicenseKey": "standard-iris.key",
      "InstanceType": "m4.xlarge",
      "OSVolumeSize": "32",
      "DataVolumeSize": "25"
   },
  ```

  注意　ディスク・サイズを変更したり、CPU を追加したりするなど、既存のノードの属性を変更すると、それらのノード (それらの永続ストレージを含む) が再作成されることがあります。この動作はクラウド・プロバイダごとにきわめて特有であり、データの破損や損失の可能性を避けるために注意して使用する必要があります。

  重要　再プロビジョニング時には、definitions.json ファイルの Label フィールドと Tag フィールドの変更はサポートされていません。

- ノードを追加するには、以下に示す 1 つ以上の方法で definitions.json ファイルを変更します。
  - 定義を追加することによって、新しいノード・タイプを追加します。例えば、データ・ノードのみを含むシャード・クラスタを導入した場合、適切な COMPUTE ノードの定義を定義ファイルに追加することによって、計算ノードを追加できます。
  - 既存のノード・タイプの定義で指定されている Count を増やすことによって、そのノード・タイプをさらに追加します。例えば、アプリケーション・サーバが既に 2 つある分散キャッシュ・クラスタにアプリケーション・サーバをさらに 2 つ追加するには、"Count": "2" を "Count": "4" に変更して AM の定義を変更します。既存のインフラストラクチャまたは導入済みの構成にノードを追加した場合、既存のノードが再起動されたり変更されたりすることはありません。また、それらの永続ストレージも変更されません。

    注釈　導入済みのシャード・クラスタにデータがロードされた後にデータ・ノードを追加する場合は、新しいサーバ間でシャード・データを自動的に再分散できます (ただし、これはクラスタをオフラインにして行う必要があります)。詳細は、"スケーラビリティ・ガイド" の "シャーディングによるデータ量に応じた InterSystems IRIS の水平方向の拡張" の章にある "データ・ノードの追加とデータの再分散" を参照してください。

    一般に、ICM では変更できず、ノードを追加した後に手動で変更しなければならないアプリケーション固有の属性が多数あります。

  - DATA、COMPUTE、AM、または WS の各ノードの定義に "LoadBalancer": "true" を追加することによって、ロード・バランサを追加します。

- ノードを削除するには、ノード・タイプの定義で指定されている Count を減らします。指定されたタイプのノードをすべて削除するには、カウントを 0 に減らします。

  注意　定義を完全に削除しないでください。Terraform で変更が検出されず、ICM が追跡しなくなった孤立したノードがインフラストラクチャまたは導入済みの構成に含まれるようになります。

  重要　1 つ以上のノードを削除する際に、削除するノードを選択することはできません。ノードは、後入れ先出し法に基づいてプロビジョニング解除されるため、最後に作成されたノードが最初に削除されます。これは特に、前回の再プロビジョニング操作で 1 つ以上のノードを追加した場合に重要です。最初にプロビジョニングされたノードよりも先にそれらのノードが削除されるためです。

  "LoadBalancer": "true" をノード定義から削除するか、値を false に変更することによって、ロード・バランサを削除することもできます。

再プロビジョニングによる既存の構成の変更には、以下のようないくつかの制限があります。

- データを格納するノード (DATA、DM、または DS) を削除することはできません。
- ミラーリングされないものからミラーリングされるもの (またはその逆) へ構成を変更することはできません。
- DATA、DS、または DM ノードは、関連する MirrorMap の設定と一致する個数でのみ、ミラー構成に追加できます ("ミラーリングの規則" を参照)。例えば、MirrorMap の既定値 primary,backup が有効な場合、DATA および DS ノードは、追加のフェイルオーバー・ペアとして構成されるよう、偶数個 (2 の倍数) でのみ追加できます。MirrorMap が primary,backup,async の場合、DATA または DS ノードは、追加の 3 メンバ・ミラーとして構成されるよう、3 の倍数の個数で追加するか、非同期なしの追加のミラーとして構成されるペアとして追加でき、DM ノードは既存の DM ミラーに現在非同期が含まれない場合のみ追加できます。
- アービター (AR ノード) をミラー構成に追加できますが、構成内のそれぞれのミラーについて、管理ポータルまたは ^MIRROR ルーチンを使用して、手動でアービターとして構成する必要があります。

既定では、icm provision コマンドを発行して既存のインフラストラクチャを変更する際に、ICM は確認のプロンプトを出します。-force オプションを使用して、スクリプトの使用時などにこのプロンプトを回避できます。

導入済みの構成を再プロビジョニングしたら、icm run コマンドを再度発行して再導入する必要があることに留意してください。

3.5.3 インフラストラクチャ管理コマンド

このセクションに示すコマンドは、ICM を使用してプロビジョニングしたインフラストラクチャの管理に使用されます。

多くの ICM コマンド・オプションは、複数のコマンドで使用できます。例えば、-role オプションは、コマンドを実行するノードのタイプを指定するために、さまざまなコマンドで使用できます。例えば、icm inventory -role AM は、導入環境内のノードのうち、タイプが AM であるもののみをリストします。また、コンテナの導入元のイメージを指定する -image オプションは、icm run コマンドと icm upgrade コマンドの両方で使用できます。ICM コマンドとオプションの完全なリストについては、"ICM リファレンス" の章にある "ICM コマンドとオプション" を参照してください。

3.5.3.1 icm inventory

icm inventory コマンドは、プロビジョニング出力の最後に、instances.json ファイルの情報に基づいて、プロビジョニングされたノードをリストします（“基本的な ICM の要素” の章の ″インスタンス・ファイル″ を参照)。次に、例を示します。

```
$ icm inventory
Machine            IP Address     DNS Name                                                  Region    Zone
-------            ----------     --------                                                  ------    ----
ANDY-DM-TEST-0001+ 00.53.183.209  ec2-00-53-183-209.us-west-1.compute.amazonaws.com  us-west-1 c
ANDY-DM-TEST-0002- 00.53.183.185  ec2-00-53-183-185.us-west-1.compute.amazonaws.com  us-west-1 c
ANDY-AM-TEST-0003  00.56.59.42    ec2-00-56-59-42.us-west-1.compute.amazonaws.com    us-west-1 c
ANDY-AM-TEST-0005  00.67.1.11     ec2-00-67-1-11.us-west-1.compute.amazonaws.com     us-west-1 c
ANDY-AM-TEST-0003  00.193.117.217 ec2-00-193-117-217.us-west-1.compute.amazonaws.com us-west-1 c
ANDY-LB-TEST-0002  (virtual AM)   ANDY-AM-TEST-1546467861.amazonaws.com              us-west-1 c
ANDY-AR-TEST-0006  00.53.201.194  ec2-00-53-201-194.us-west-1.compute.amazonaws.com  us-west-1 c
```

注釈　ミラーリングされたノードが構成に含まれている場合、初期のミラー・フェイルオーバーの割り当ては、上の例のように、対象の各プライマリのマシン名の後ろにある + (プラス) と、対象の各バックアップのマシン名の後ろにある – (マイナス) で示されます。ただし、これらの割り当ては変更されることがあります。導入後に、icm ps コマンドを使用して、導入されたノードのミラー・メンバ・ステータスを表示します。

–machine オプションまたは –role オプションを使用して、ノード名別またはロール別にフィルタすることもできます。例えば、前の例で同じクラスタでフィルタした場合、以下のようになります。

```
$ icm inventory -role AM
Machine            IP Address     DNS Name                                                  Region    Zone
-------            ----------     --------                                                  ------    ----
ANDY-AM-TEST-0003 00.56.59.42    ec2-00-56-59-42.us-west-1.compute.amazonaws.com    us-west-1 c
ANDY-AM-TEST-0005 00.67.1.11     ec2-00-67-1-11.us-west-1.compute.amazonaws.com     us-west-1 c
ANDY-AM-TEST-0003 00.193.117.217 ec2-00-193-117-217.us-west-1.compute.amazonaws.com us-west-1 c
```

クラウド・プロバイダから提供された完全修飾 DNS 名の幅が広すぎて読みにくい場合は、Docker の wide 引数を指定して –options オプションを使用すると、出力の幅を広げることができます。以下に例を示します。

```
icm inventory -options wide
```

–options オプションの詳細は、″カスタム・コンテナおよびサードパーティ・コンテナでの ICM の使用″ を参照してください。

3.5.3.2 icm ssh

icm ssh コマンドは、指定されたホスト・ノード上で任意のコマンドを実行します。複数のコマンドからの出力を混合すると解釈が難しくなるため、出力はファイルに書き込まれ、出力ファイルのリストが提供されます。例を以下に示します。

```
$ icm ssh -command "ping -c 5 intersystems.com" -role DM
Executing command 'ping -c 5 intersystems.com' on ANDY-DM-TEST-0001...
Executing command 'ping -c 5 intersystems.com' on ANDY-DM-TEST-0002...
...output in ./state/ANDY-DM-TEST/ANDY-DM-TEST-0001/ssh.out
...output in ./state/ANDY-DM-TEST/ANDY-DM-TEST-0002/ssh.out
```

ただし、以下に示すように、–machine オプションや –role オプションを使用してノードを 1 つだけ指定した場合、またはノードが 1 つしかない場合は、出力はコンソールにも書き込まれます。

```
$ icm ssh -command "df -k" -machine ANDY-DM-TEST-0001
Executing command 'df -k' on ANDY-DM-TEST-0001...
...output in ./state/ANDY-DM-TEST/ANDY-DM-TEST-0001/ssh.out

Filesystem     1K-blocks    Used Available Use% Mounted on
rootfs          10474496 2205468   8269028  22% /
tmpfs            3874116       0   3874116   0% /dev
tmpfs            3874116       0   3874116   0% /sys/fs/cgroup
/dev/xvda2      33542124 3766604  29775520  12% /host
/dev/xvdb       10190100   36888   9612540   1% /irissys/data
/dev/xvdc       10190100   36888   9612540   1% /irissys/wij
/dev/xvdd       10190100   36888   9612540   1% /irissys/journal1
/dev/xvde       10190100   36888   9612540   1% /irissys/journal2
shm                65536     492     65044   1% /dev/shm
```

また、インタラクティブ・モードで icm ssh コマンドを使用して、長時間実行されるコマンドや、ブロック・コマンド、またはインタラクティブ・コマンドをホスト・ノード上で実行することもできます。単一ノードの導入環境でコマンドを実行する場合を除き、-interactive フラグと一緒に、コマンドを単一のノードに限定する -role オプションまたは -machine オプションを指定する必要があります。-command オプションを使用しない場合は、宛先ユーザの既定シェル (例えば bash) が起動されます。

コマンドをインタラクティブに実行する例については、"icm exec" を参照してください。

注釈　"サービス管理コマンド" で説明されている 2 つのコマンド icm exec (指定されたコンテナに対して任意のコマンドを実行する) と icm session (指定されたノード上で InterSystems IRIS インスタンスのインタラクティブ・セッションを開く) は、icm ssh と組み合わせて、ICM の導入を操作するための強力なツールのセットとして使用できます。ローカル ICM コンテナから指定ノードのホスト OS にファイルまたはディレクトリを安全にコピーする icm scp コマンドは、多くの場合、icm ssh と共に使用します。

3.5.3.3 icm scp

icm scp コマンドは、ローカル ICM コンテナから指定ノードのローカル・ファイル・システムに、ファイルまたはディレクトリを安全にコピーします。コマンドの構文は以下のとおりです。

```
icm scp -localPath local-path [-remotePath remote-path]
```

localPath と remotePath の両方とも、ファイルまたはディレクトリのどちらも指定できます。remotePath がディレクトリの場合は、末尾にスラッシュ (/) を付ける必要があり、そうでなければファイルと見なされます。両方がディレクトリの場合は、ローカル・ディレクトリの内容が再帰的にコピーされます。ディレクトリ自体をコピーする場合は、localPath から先頭のスラッシュ (/) を削除してください。

SSHUser フィールドで指定するユーザは、オプションの remote-path 引数で指定するホスト・ファイル・システムの場所に対して必要な権限を持っている必要があります。remote-path の既定値は、ホスト OS で定義されている **$HOME** です。

注釈　icm cp コマンドも参照してください。このコマンドは、指定のノードから指定のコンテナに、またはコンテナからローカル・ファイル・システムに、ローカル・ファイルまたはディレクトリをコピーします。

3.6 サービスの導入と管理

ICM は、Docker イメージを使用してソフトウェア・サービスの導入を行い、Docker を呼び出すことによってイメージをコンテナとして実行します。イメージを使用してコンテナ化された導入により、操作の容易性と DevOps への適合がサポートされ、手動アップグレードのリスクを回避できます。Docker のほかに、ICM は JDBC 上で InterSystems IRIS 固有の構成も行います。

さまざまなコンテナ管理および調整ツールが利用でき、これらのツールを使用して ICM の導入機能と管理機能を拡張できます。

- icm run コマンド
- サービスの再導入
- コンテナ管理コマンド
- サービス管理コマンド

3.6.1 icm run コマンド

icm run コマンドは、プロビジョニングされたそれぞれのノード上で、指定されたイメージからコンテナをプル、作成、および起動します。既定では、構成ファイルの DockerImage フィールドで指定されたイメージが使用され、導入されるコンテナの名前は **iris** です。この名前は、以下のインターシステムズのイメージ (またはこれらのイメージに基づくイメージ) から作成されたコンテナ向けに予約済みで、そのコンテナでのみ使用する必要があります。これらのイメージは、"InterSystems Container Registry の使用" で説明されているように、InterSystems Container Registry から利用できます。

- **iris** — InterSystems IRIS のインスタンスを含みます。

  インターシステムズが配布している InterSystems IRIS イメージと、InterSystems IRIS ベースのアプリケーションを含むカスタム・イメージのベースとしてそれらのイメージを使用する方法については、"InterSystems IRIS コンテナの実行" で詳しく説明しています。

  InterSystems IRIS イメージは、ICM によって DATA、COMPUTE、DM、AM、DS、および QS の各ノードとして導入されます。**iris** イメージを導入する場合、導入する **iris** コンテナすべてについて 1 つ以上の InterSystems IRIS 構成設定をオーバーライドすることも、それぞれのノード・タイプに導入されるコンテナについて異なる設定をオーバーライドすることもできます。詳細は、"カスタマイズされた InterSystems IRIS 構成を使用した導入" を参照してください。

- **Iris-lockeddown** — ロック・ダウン・セキュリティを使用してインストールされた InterSystems IRIS のインスタンスを含みます。このイメージを使用すると、きわめて安全な InterSystems IRIS コンテナを導入することによって最も厳格なセキュリティ要件をサポートできます。このイメージのコンテナと標準の **iris** イメージのコンテナとの違いは、"InterSystems IRIS ロック・ダウン・コンテナ" を参照してください。

  重要　**iris-lockeddown** イメージを使用して InterSystems IRIS コンテナを導入する前に、必ずこのイメージのドキュメントを確認してください。

  iris-lockeddown イメージから導入した InterSystems IRIS コンテナでは、インスタンスの Web サーバが無効になるため、管理ポータルも無効になります。このようなコンテナで管理ポータルを有効にし、このセクションの最後で説明しているように管理ポータルを使用して導入環境に接続できるようにするには、値 1 を指定した webserver プロパティ (webserver=1) を、UserCPF プロパティで指定した構成マージ・ファイルに追加します ("カスタマイズされた InterSystems IRIS 構成を使用した導入" を参照してください)。

  注釈　追加のセキュリティ対策として、コンテナレス・モードでは、InterSystems IRIS の非 root インスタンス、つまり root 特権のないユーザがインストールして所有するインスタンスを ICM からインストールできます。

- **iris-ml** — IntergratedML 機能を備えた InterSystems IRIS のインスタンスを含みます。IntegratedML は、自動機械学習機能を SQL から直接使用して予測モデルを作成および使用できる InterSystems IRIS の機能です。

- **irishealth**、**irishealth-lockeddown**、**irishealth-ml** — InterSystems IRIS for Health® のインスタンスを含みます。InterSystems IRIS for Health は、InterSystems IRIS を中心として構築した機能完備の医療向けプラットフォームであり、豊富なデータを扱うミッション・クリティカルな医療向けアプリケーションの迅速な開発および導入を可能にします。ロック・ダウンおよび IntegratedML は、前述のように InterSystems IRIS のオプションです。InterSystems IRIS for Health イメージは、InterSystems IRIS イメージと同様の方法で ICM によって導入されます。

- **webgateway**, **webgateway-lockeddown**, **webgateway-nginx** — Apache Web サーバまたは Nginx Web サーバと共にインターシステムズの Web ゲートウェイ・インスタンスを含みます。**webgateway-lockeddown** イメージは、ロック・ダウン InterSystems IRIS イメージに似ています。これらのイメージの詳細は、"コンテナ内でのインターシステムズ製品の実行" の "InterSystems Web ゲートウェイ・コンテナの使用法" を参照してください。**webgateway** イメージは ICM により WS ノードとして導入されます。このノードは、ノードレベルのシャード・クラスタでは DATA ノードの Web サーバまたは DATA ノードと COMPUTE ノードの Web サーバとして構成され、その他の構成では AM ノードまたは DM ノードの Web サーバとして構成されます。

- **arbiter** – ミラー・アービターとして動作する ISCAgent インスタンスを含みます。このイメージの使用の詳細は、"InterSystems IRIS コンテナを使用したミラーリング" を参照してください。**arbiter** イメージは AR ノードに導入されます。このノードは、ミラーリングされた導入のアービター・ホストとして構成されます。ミラーリングされた導入とトポロジの詳細は、"ICM クラスタのトポロジ" を参照してください。
- **iam** – ICM が CN ノードとして導入する InterSystems API Manager を含みます。詳細は、"ICM リファレンス" の章の "InterSystems API Manager の導入" を参照してください。
- **sam** – ICM が SAM ノードとして導入する System Alerting and Monitoring (SAM) クラスタ・モニタリング・ソリューションの SAM マネージャ・コンポーネントを含みます。詳細は、"ICM リファレンス" の章の "ICM でのモニタリング" を参照してください。

注釈　上記リストのイメージのうち、**sam** を除くすべてのイメージを ARM プラットフォームで利用できます (**iris-arm64** など)。

definitions.json ファイルの各ノードの定義に DockerImage フィールドを追加すると、それぞれのノード・タイプで異なる InterSystems IRIS イメージを実行できます。例えば、AR ノードで **arbiter** イメージを、WS ノードで **webgateway** イメージを、その他のノードで **iris** イメージを実行するためにはこれを実行する必要があります。ノード・タイプおよび対応するインターシステムズのイメージのリストについては、"ICM リファレンス" の章にある "ICM ノード・タイプ" を参照してください。

重要　DockerImage フィールドまたは icm run コマンドの -image オプションでノードについて誤ったインターシステムズのイメージが指定された場合 (例えば、**iris** イメージが AR (アービター) ノードについて指定された場合や、インターシステムズのいずれかのイメージが CN ノードについて指定された場合)、導入は失敗し、ICM から該当するメッセージが返されます。したがって、defaults.json ファイルの DockerImage フィールドで **iris** イメージを指定して、definitions.json ファイルに AR 定義または WS 定義を含める場合は、AR 定義または WS 定義に DockerImage フィールドを含めて既定値をオーバーライドし、適切なイメージ (それぞれ **arbiter** または **webgateway**) を指定する必要があります。

ICM を起動したイメージと DockerImage フィールドを使用して指定したインターシステムズのイメージのメジャー・バージョンが一致している必要があります。例えば、2022.1 バージョンの ICM イメージを使用して 2022.2 バージョンの InterSystems IRIS イメージを導入することはできません。インターシステムズのコンテナをアップグレードする前に ICM をアップグレードする方法については、付録 "分散管理モードでの導入環境の共有" の "分散管理モードを使用した ICM のアップグレード" を参照してください。

注釈　インターシステムズのコンテナ・イメージは Open Container Initiative (OCI) の仕様に準拠しており、Docker Enterprise Edition エンジンを使用して構築されます。このエンジンは、OCI 標準を全面的にサポートしており、これによってイメージは認定され、Docker Hub レジストリで公開することができます。

インターシステムズのイメージは、広く知られたコンテナ向け Ubuntu オペレーティング・システムを使用して構築およびテストされているため、オンプレミスとパブリック・クラウドの両方において、Linux ベースのオペレーティング・システム上の OCI に準拠するすべてのランタイム・エンジンでサポートされます。

コマンド行で InterSystems IRIS コンテナをすばやく実行する方法は、"InterSystems IRIS の基礎：InterSystems IRIS コンテナの実行" を参照してください。ICM 以外の方法を使用してコンテナに InterSystems IRIS および InterSystems IRIS ベースのアプリケーションを導入する方法は、"コンテナ内でのインターシステムズ製品の実行" を参照してください。

また、icm run で -image と -container のコマンド行オプションを使用して、別のイメージとコンテナ名を指定することもできます。これにより、icm run コマンドを複数回使用して、複数のイメージから作成された複数のコンテナをプロビジョニングされた各ノードに導入することができます。最初は、前の段落で説明したように、ノード定義の DockerImage フィールドで指定されたイメージを実行し、各ノードに **iris** コンテナ (1 つのみ) を導入します。それ以降は、-image オプションと -container オプションを使用して、すべてまたは一部のノードでカスタム・イメージを実行します。特定のノードで実行されるコンテナには、それぞれ一意の名前が必要です。また、-machine オプションと -role オプションを使用して、コン

テナ導入を特定ノードまたは特定タイプのノードに制限することもできます (例えば、プロビジョニングされた特定ノードにユーザ独自カスタム・コンテナを導入する場合)。

もう 1 つのよく使用されるオプション -iscPassword は、導入されるすべての InterSystems IRIS コンテナに対して設定する InterSystems IRIS パスワードを指定します。この値は構成ファイルに含めることもできますが、コマンド行オプションを使用すれば、パスワードを平文のレコードに保管せずに済みます。InterSystems IRIS パスワードがどちらの方法でも指定されない場合は、ICM からパスワードの入力が求められます (入力内容はマスクされます)。

注釈　セキュリティ上の理由から、ICM では InterSystems IRIS パスワード (何らかの方法で指定されたもの) をプレーン・テキストで送信する代わりに、暗号化ハッシュ関数を使用して、ハッシュ化されたパスワードとソルトをローカルで生成し、ホスト・ノードに導入されている InterSystems IRIS コンテナに SSH を使用して送信します。

前述の内容すべてを前提として、icm run コマンドを使用したコンテナの導入について、以下の 3 つの例を考えてみましょう。(これらは完全な手順を示すものではなく、特定のノードに特定のコンテナを導入する作業に関連する手順の要素に限って説明します。)

- 1 つの DM ノードといくつかの AM ノードを含む分散キャッシュ・クラスタを導入するには、以下の手順で行います。

  1. defaults.json ファイルを作成する際に、"構成ファイル、状態ファイル、およびログ・ファイル" と "導入の定義" で説明されているように、以下の項目を組み込んで **iris** コンテナの作成に使用する既定のイメージを指定します。

     ```
     "DockerImage": "intersystems/iris:latest-em"
     ```

  2. ICM コマンド行で以下のコマンドを実行します。

     ```
     icm run -iscPassword "<password>"
     ```

     指定の初期パスワードが設定された InterSystems IRIS インスタンスを含む **iris** という名前のコンテナが、それぞれのノードに導入されます。コンテナの導入後、ICM は必要な ECP 構成を実行します。

- ミラーリングされた DATA ノードと AR (アービター) ノードを含む基本シャード・クラスタを導入するには、以下の手順で行います。

  1. defaults.json ファイルを作成する際に、"構成ファイル、状態ファイル、およびログ・ファイル" と "導入の定義" で説明されているように、以下の項目を組み込んで **iris** コンテナの作成に使用する既定のイメージを指定し、ミラーリングを有効にします ("ミラーリングの規則" を参照)。

     ```
     "DockerImage": "intersystems/iris:latest-em",
     "Mirror": "true"
     ```

  2. definitions.json ファイルを作成する際に、AR ノード定義で **arbiter** イメージを指定することによって、AR ノードのみを対象に既定値ファイル内の DockerImage フィールドをオーバーライドします。例を以下に示します。

     ```
     {
          "Role": "AR",
          "Count": "1",
          "DockerImage": "intersystems/arbiter:latest-em"
      }
     ```

  3. ICM コマンド行で以下のコマンドを実行します。

     ```
     icm run -iscPassword "<password>"
     ```

     指定の初期パスワードが設定された InterSystems IRIS インスタンスを含む、**iris** という名前のコンテナがそれぞれの DATA ノードに導入されます。ミラー・アービターとして動作する ISCAgent を含む、**iris** という名前のコンテナが AR ノードに導入されます。ICM は、必要なシャーディングおよびミラーリング構成をコンテナの導入後に実行します。

- **iris** コンテナのスタンドアロン InterSystems Iris インスタンスと追加コンテナ (カスタム・イメージから作成されます) を含む DM ノードといくつかの DM 接続 WS ノードを導入するには、以下の手順で行います。

  1. definitions.json ファイルを作成する際に、"構成ファイル、状態ファイル、およびログ・ファイル" および "導入の定義" で説明されているように、DM ノードに対して **iris** イメージを指定し、WS ノードに対して **webgateway** イメージを指定します。以下に例を示します。

     ```
     {
          "Role": "DM",
          "Count": "1",
          "DockerImage": "intersystems/iris-arm64:latest-em"
      },
      {
          "Role": "WS",
          "Count": "3",
          "DockerImage": "intersystems/webgateway:latest-em"
      }
     ```

  2. ICM コマンド行で以下のコマンドを実行します。

     ```
     icm run
     ```

     ICM から InterSystems IRIS の初期パスワードの入力が求められ (入力内容はマスクされます)、InterSystems IRIS インスタンスを含む、**iris** という名前のコンテナが DM ノードに導入されます。また、インターシステムズの Web ゲートウェイのインストールと Apache Web サーバを含む、**iris** という名前のコンテナが各 WS ノードに導入され、コンテナの導入後に ICM によって必要な Web サーバの構成が実行されます。

  3. 別の icm run コマンドを実行して、カスタム・コンテナを DM ノードに導入します。例えば、以下のいずれかを実行します。

     ```
     icm run -container customsensors -image myrepo/sensors:1.0 -role DM
     ```

     ```
     icm run -container customsensors -image myrepo/sensors:1.0 -machine ANDY-DM-TEST-0001
     ```

     リポジトリ内のイメージ sensors から作成された customsensors という名前のコンテナが DM ノードに導入されます。

以下の詳細な考慮事項に注意してください。

- コンテナ名 **iris** は、すべての ICM コンテナおよびサービス管理コマンドの既定値です (以降のセクションで説明するとおり)。このため、別の名前を使用して導入した追加のコンテナに関連するコマンドを実行する場合は、-container オプションを使用してその名前を明示的に参照する必要があります。例えば、最後に示した例のカスタム・コンテナを DM ノードから削除するには、以下のコマンドを発行します。

  ```
  icm rm -container customsensors -machine ANDY-DM-TEST-0001
  ```

  -container customsensors を指定しなければ、このコマンドは既定で **iris** コンテナを削除します。

- DockerRegistry、DockerUsername、および DockerPassword の各フィールドは、指定したイメージが配置されている Docker レジストリを指定し、ログインする (プライベートの場合) ために必要です。詳細は、"Docker リポジトリ" を参照してください。

- icm run コマンドで -namespace コマンド行オプションを使用して、既定値ファイルで指定されたネームスペース (既定値ファイルで指定されていない場合は既定値の IRISCLUSTER) をオーバーライドした場合、instances.json ファイル ("基本的な ICM の要素" の章にある "インスタンス・ファイル" を参照) 内の Namespace フィールドの値が、指定した名前で更新され、これが、icm session コマンドおよび icm sql コマンドを使用する際の既定のネームスペースになります。

-options オプションを使用して、--volume など、追加の Docker オプションを icm run コマンド行に指定できます。例を以下に示します。

```
icm run -options "--volume /shared:/host" image intersystems/iris:latest-em
```

-options オプションの詳細は、"カスタム・コンテナおよびサードパーティ・コンテナでの ICM の使用" を参照してください。

-command オプションを icm run で使用して、Docker エントリ・ポイントの引数 (またはエントリ・ポイントの代わり) を指定できます。詳細は、"既定のコマンドのオーバーライド" を参照してください。

ICM は複数のスレッドで Docker コマンドを実行するので、コンテナが各ノードに導入される順序は不定です。これを以下の例に示します。この例は、"導入の定義" で説明されているシャード・クラスタ構成の導入からの出力を表しています。繰り返しの行は簡略化のために省いています。

```
$ icm run
Executing command 'docker login' on ANDY-DATA-TEST-0001...
...output in /Samples/AWS/state/ANDY-DATA-TEST/ANDY-DATA-TEST-0001/docker.out
...
Pulling image intersystems/iris:latest-em on ANDY-DATA-TEST-0001...
...pulled ANDY-DATA-TEST-0001 image intersystems/iris:latest-em
...
Creating container iris on ANDY-DATA-TEST-0002...
...
Copying license directory /Samples/license/ to ANDY-DATA-TEST-0003...
...
Starting container iris on ANDY-DATA-TEST-0004...
...
Waiting for InterSystems IRIS to start on ANDY-DATA-TEST-0002...
...
Configuring SSL on ANDY-DATA-TEST-0001...
...
Enabling ECP on ANDY-DATA-TEST-0003...
...
Setting System Mode on ANDY-DATA-TEST-0002...
...
Acquiring license on ANDY-DATA-TEST-0002...
...
Enabling shard service on ANDY-DATA-TEST-0001...
...
Assigning shards on ANDY-DATA-TEST-0001...
...
Configuring application server on ANDY-DATA-TEST-0003...
...
Management Portal available at:
http://ec2-00-56-140-23.us-west-1.compute.amazonaws.com:52773/csp/sys/UtilHome.csp
```

実行を完了した ICM は、DM ノード上の **iris** コンテナで実行中の InterSystems IRIS インスタンスの管理ポータルへのリンクを出力します。また、シャード・クラスタが導入されている場合は、データ・ノード 1 上のインスタンスの管理ポータルへのリンクを出力します。このノード 1 は番号が最も小さいノードであり、この場合は ANDY-DATA-TEST-001 になります。この DM ノードまたはシャード・クラスタをミラーリングしている場合、上記のリンクは初期プライマリに対するリンクとなります。ただし、ミラーリングした DM ノードまたは DATA ノードにロード・バランサ (ロール LB) を定義している場合、このリンクはミラー認識ロード・バランサを参照し、必ず現在のプライマリへのリンクとなります。

3.6.2 サービスの再導入

導入プロセスの柔軟性と回復力を可能な限り高めるために、icm run コマンドは完全に再入可能であり、同じ導入環境について複数回発行できます。icm run コマンドが繰り返されると、ICM は、影響を受けるコンテナを停止して削除し (icm stop コマンドと icm rm コマンドに相当)、その後、初期導入パスの一環として作成してマウントした InterSystems IRIS インスタンス固有のデータのストレージ・ボリュームを保持したまま (“ICM リファレンス” の章にある "ICM によってマウントされるストレージ・ボリューム" を参照)、該当するイメージからコンテナを再度作成して起動します。

icm run コマンドを複数回実行してサービスを再導入する主な理由としては、以下の 4 つがあります。

- 既存のコンテナを既存のストレージ・ボリュームと共に再導入する。

影響を受ける InterSystems IRIS コンテナのインスタンス固有のストレージ・ボリュームを保持したまま、導入済みのコンテナを新しいバージョンに置き換え、それによって既存のインスタンスを再導入するには、最初にコンテナを導入した元の icm run コマンドを繰り返します。LicenseDir フィールドで指定されたディレクトリ内のライセンスを更新した場合など、再導入を必要とする変更を定義ファイル内で行った場合に、この作業を行うことがあります。

- InterSystems IRIS コンテナを既存のストレージ・ボリュームなしで再導入する。

  インスタンス固有のストレージ・ボリュームを保持せずに、導入環境内の InterSystems IRIS コンテナを置き換える場合、以下のコマンドを使用して、再導入の前にそれらのインスタンスについてそのデータを削除することができます。

  ```
  icm ssh -command "sudo rm -rf /<mount_dir>/*/*"
  ```

  mount_dir は、InterSystems IRIS データ・ディレクトリ、WIJ ディレクトリ、およびジャーナル・ディレクトリがマウントされているディレクトリです (既定では /irissys/ です。または DataMountPoint、WIJMountPoint、Journal1MountPoint、および Journal2MountPoint の各フィールドで構成されたディレクトリです。詳細は、“ICM リファレンス” の章にある ″ICM によってマウントされるストレージ・ボリューム″ を参照してください)。必要に応じて、-role オプションまたは -machine オプションを使用して、このコマンドを特定のノードに限定することもできます。その後、最初に InterSystems IRIS コンテナを導入した icm run コマンドを繰り返すと、インスタンス固有のボリュームを引き続き保持しているものは同じインスタンスとして再導入され、ボリュームを削除したものは新しいインスタンスとして再導入されます。

- ″インフラストラクチャの再プロビジョニング″ の説明に従ってインフラストラクチャに追加したノードにサービスを導入する。

  インフラストラクチャにノードを追加した後に icm run コマンドを繰り返すと、既存のノード上のコンテナは前述のように再導入され (ストレージ・ボリュームと共に再導入されるか、削除した場合はストレージ・ボリュームなしで再導入される)、新しいノードには新しいコンテナが導入されます。これにより、必要に応じて、既存のノードを新しい導入トポロジに合わせて再構成することができます。

- 導入エラーを解決する。

  ネットワークの遅延や切断、クラウド・プロバイダ・サービスの中断など (エラー・ログ・メッセージによって示される)、ICM の制御下にない要因によって 1 つ以上のノードで icm run コマンドが失敗した場合、コマンドを再度発行できます。ほとんどの場合、繰り返し試行することで導入が正常に完了します。どうしてもエラーが解決されず、手動による操作が必要な場合は (いずれかの構成ファイル内のエラーが原因である場合など)、問題を修正した後、icm run を再発行する前に、前述の説明に従って、影響を受けるノードのストレージ・ボリュームを削除しなければならないことがあります。ICM は、インスタンス固有のデータがないノードを新しいノードとして認識し、すべての構成が正常に完了した場合にのみ、InterSystems IRIS コンテナのストレージ・ボリュームを完全に導入されたものとしてマークするためです。構成が開始されたものの正常に完了しておらず、ボリュームがマークされていない場合、ICM はそのノードで再導入を行うことはできません。新しい導入では、-role や -machine の制約を指定せずにコマンド icm ssh -command ″sudo rm -rf /irissys/*/*″ を発行して、InterSystems IRIS を再導入するノードをすべてロールバックするのが最も簡単な方法である場合があります。

3.6.3 コンテナ管理コマンド

このセクションに示すコマンドは、プロビジョニングしたインフラストラクチャ上で導入したコンテナの管理に使用されます。

多くの ICM コマンド・オプションは、複数のコマンドで使用できます。例えば、-role オプションは、コマンドを実行するノードのタイプを指定するために、さまざまなコマンドで使用できます。例えば、icm inventory -role AM は、導入環境内のノードのうち、タイプが AM であるもののみをリストします。また、コンテナの導入元のイメージを指定する -image オプションは、icm run コマンドと icm upgrade コマンドの両方で使用できます。ICM コマンドとオプションの完全なリストについては、“ICM リファレンス” の章にある ″ICM コマンドとオプション″ を参照してください。

3.6.3.1 icm ps

導入の完了時に、icm ps コマンドはノード上で実行されているコンテナの実行状態を表示します。例を以下に示します。

```
$ icm ps -container iris
Machine             IP Address    Container Status Health  Image
-------             ----------    --------- ------ ------  -----
ANDY-DATA-TEST-0001 00.56.140.23  iris      Up     healthy intersystems/iris:latest-em
ANDY-DATA-TEST-0002 00.53.190.37  iris      Up     healthy intersystems/iris:latest-em
ANDY-DATA-TEST-0003 00.67.116.202 iris      Up     healthy intersystems/iris:latest-em
ANDY-DATA-TEST-0004 00.153.49.109 iris      Up     healthy intersystems/iris:latest-em
```

-container の制約を省略すると、ノード上で実行中のすべてのコンテナがリストされます。これには、ICM によって導入された他のコンテナ（例えば、Weave ネットワーク・コンテナや、icm run コマンドを使用して導入したカスタム・コンテナまたはサードパーティ・コンテナなど）と、ICM の導入完了後にその他の方法で導入したあらゆるコンテナの両方が含まれます。

ノード名、IP アドレス、コンテナ名、およびコンテナの作成元のイメージのほか、icm ps コマンドには、以下の列が含まれています。

- Status ― Docker によって生成されるステータス値の 1 つ：created、restarting、running、removing（または up）、paused、exited、または dead。
- Health ― **iris** コンテナ、**arbiter** コンテナ、および **webgateway** コンテナの場合、starting、healthy、または unhealthy のいずれかの値。その他のコンテナの場合は none（または空白）。Status が exited の場合、Health にその終了値が表示される場合があります（0 は成功を意味します）。

  iris コンテナの場合、Health の値にはコンテナ内の InterSystems IRIS インスタンスのヘルス状態が反映されます（InterSystems IRIS のヘルス状態の詳細は、"監視ガイド" の "システム・モニタの使用" の章にある "システム・モニタのヘルス状態" を参照してください）。**arbiter** コンテナの場合は、ISCAgent のステータスが、**webgateway** コンテナの場合は、インターシステムズの Web ゲートウェイの Web サーバのステータスが反映されます。unhealthy は一時的な可能性があることに留意してください。これはその後クリアされる警告に起因している場合があるからです。
- Mirror ― ミラーリングが有効になっている場合（"ミラーリングの規則" を参照）、ミラー・メンバのステータス（PRIMARY、BACKUP、SYNCHRONIZING など）が、%SYSTEM.Mirror.GetMemberStatus() ミラーリング API 呼び出しによって返されます。次に、例を示します。

  ```
  $ icm ps -container iris
  Machine             IP Address    Container Status Health  Mirror  Image
  -------             ----------    --------- ------ ------  ------  -----
  ANDY-DATA-TEST-0001 00.56.140.23  iris      Up     healthy PRIMARY intersystems/iris:latest-em
  ANDY-DATA-TEST-0002 00.53.190.37  iris      Up     healthy BACKUP  intersystems/iris:latest-em
  ANDY-DATA-TEST-0003 00.67.116.202 iris      Up     healthy PRIMARY intersystems/iris:latest-em
  ANDY-DATA-TEST-0004 00.153.49.109 iris      Up     healthy BACKUP  intersystems/iris:latest-em
  ```

  各ステータスが意味することの説明は、"高可用性ガイド" の “ミラーリング” の章の "ミラー・メンバのジャーナル転送およびデジャーナリングのステータス" を参照してください。

以下には、その他の導入および管理フェーズのコマンドがリストされています。これらのコマンドの詳細は、"ICM リファレンス" を参照してください。

3.6.3.2 icm stop

icm stop コマンドは、指定コンテナ（または、既定値の **iris**）を指定ノード（また、マシンまたはロールの制約が指定されない場合、すべてのノード）で停止します。例えば、分散キャッシュ・クラスタ構成内のアプリケーション・サーバ上で InterSystems IRIS コンテナを停止するには、以下のように入力します。

```
$ icm stop -container iris -role DS

Stopping container iris on ANDY-DATA-TEST-0001...
Stopping container iris on ANDY-DATA-TEST-0002...
Stopping container iris on ANDY-DATA-TEST-0004...
Stopping container iris on ANDY-DATA-TEST-0003...
...completed stop of container iris on ANDY-DATA-TEST-0004
...completed stop of container iris on ANDY-DATA-TEST-0001
...completed stop of container iris on ANDY-DATA-TEST-0002
...completed stop of container iris on ANDY-DATA-TEST-0003
```

3.6.3.3 icm start

icm start コマンドは、指定コンテナ（または、既定値の **iris**）を指定ノード（また、マシンまたはロールの制約が指定されない場合、すべてのノード）で起動します。例えば、停止したアプリケーション・サーバ InterSystems IRIS コンテナの 1 つを再起動するには、以下のように入力します。

```
$ icm start -container iris -machine ANDY-DATA-TEST-0002...
Starting container iris on ANDY-DATA-TEST-0002...
...completed start of container iris on ANDY-DATA-0002
```

3.6.3.4 icm pull

icm pull コマンドは、指定イメージを指定マシンにダウンロードします。例えば、シャード・クラスタ内のデータ・ノード 1 にイメージを追加するには、以下のように入力します。

```
$ icm pull -image intersystems/webgateway:latest-em -role DATA
Pulling ANDY-DATA-TEST-0001 image intersystems/webgateway:latest-em...
...pulled ANDY-DATA-TEST-0001 image intersystems/webgateway:latest-em
```

プルするイメージが、定義ファイルの DockerImage フィールドで指定されたものであれば、-image オプションは必要ないことに注意してください。例を以下に示します。

```
"DockerImage": "intersystems/iris-arm64:latest-em",
```

icm run コマンドは、ホストにまだ存在しないイメージを自動的にプルしますが、テストやステージングなどの目的には明示的に icm pull を指定する必要が生じることがあります。

3.6.3.5 icm rm

icm rm コマンドは、指定されたノードから、またはすべてのノードから（マシンまたはロールが指定されていない場合）、指定されたコンテナ（または既定値の **iris**）を削除します。ただし、起動された元のイメージは削除しません。停止しているコンテナのみを削除できます。

3.6.3.6 icm upgrade

icm upgrade コマンドは、指定マシンで指定コンテナを置き換えます。ICM は、以下のイベントのシーケンスを調整して、アップグレードを実行します。

1. 新規イメージをプルする
2. 新規コンテナを作成する
3. 既存のコンテナを停止する

4. 既存のコンテナを削除する
5. 新規コンテナを起動する

ステップ 1 と 2 で新規イメージのステージングを行うことにより、ステップ 3 から 5 の間に必要なダウンタイムが比較的短時間に抑えられます。

例えば、アプリケーション・サーバ上の InterSystems IRIS コンテナをアップグレードするには、以下のように入力します。

```
$ icm upgrade -image intersystems/iris:latest-em -machine ANDY-AM-TEST-0003
Pulling ANDY-AM-TEST-0003 image intersystems/iris:latest-em...
...pulled ANDY-AM-TEST-0003 image intersystems/iris:latest-em
Stopping container ANDY-AM-TEST-0003...
...completed stop of container ANDY-AM-TEST-0003
Removing container ANDY-AM-TEST-0003...
...removed container ANDY-AM-TEST-0003
Running image intersystems/iris:latest-em in container ANDY-AM-TEST-0003...
...running image intersystems/iris:latest-em in container ANDY-AM-TEST-0003
```

icm upgrade コマンドには -image オプションは任意です。イメージを指定しないと、ICM では instances.json ファイルの DockerImage フィールドの値が使用されます (″“基本的な ICM の要素”″ の ″インスタンス・ファイル″ の章を参照してください)。イメージを指定すると、アップグレードの完了時に、その値が指定のイメージで更新されます。

注釈　ICM の起動元のイメージと導入するインターシステムズのイメージのメジャー・バージョンが一致している必要があります。例えば、2022.1 バージョンの ICM を使用して 2022.2 バージョンの InterSystems IRIS を導入することはできません。そのため、インターシステムズのコンテナをアップグレードする前に ICM をアップグレードする必要があります。

iris 以外のコンテナをアップグレードする場合は、-container オプションを使用してコンテナ名を指定する必要があります。

InterSystems IRIS コンテナのアップグレードに関する重要な情報については、″コンテナ内でのインターシステムズ製品の実行″ の ″InterSystems IRIS コンテナのアップグレード″ を参照してください。

ICM 全体に言えることですが、icm upgrade コマンドはスクリプト作成に使用されることを意図しているので、さまざまなアップグレードのタイプに応じてスクリプト化した異なる手順を作成できます。スクリプト化しておけば、その手順を毎回手動で繰り返す必要がありません。例えば、インストール・ガイドの ″ミラーのアップグレード″ にあるように、ミラーのメンバのアップグレードは、固有の手順を使用して固有の順序で実行する必要があります。icm upgrade コマンドによるスクリプト作成は、この作業に効果的です。例えば、以下のような基本的手順のいくつかを組み込んだプロシージャを作成できます。

1. icm ps コマンドを使用して、ミラーリングしたシャード・クラスタで現在のプライマリ・フェイルオーバー・メンバを特定します。

   ```
   $ icm ps -container iris
   Machine IP Address Container Status Health Mirror Image
   ------- ---------- --------- ------ ------ ------ -----
   Andy-DATA-TEST-0001  35.229.124.197 iris Up healthy PRIMARY intersystems/iris:2022.1.0.205.0
   Andy-DATA-TEST-0002  34.138.234.124 iris Up healthy BACKUP  intersystems/iris:2022.1.0.205.0
   Andy-DATA-TEST-0003  34.139.3.48    iris Up healthy PRIMARY intersystems/iris:2022.1.0.205.0
   Andy-DATA-TEST-0004  34.73.107.58   iris Up healthy BACKUP  intersystems/iris:2022.1.0.205.0
   ```

2. icm upgrade で -machine オプションを使用して、最初にアップグレードするバックアップを選択してアップグレードします。

   ```
   icm upgrade -machine ".*(2|4)" -image intersystems/iris:2022.1.0.223.0
   ```

3. icm upgrade -machine を使用してプライマリをアップグレードし、以前にアップグレードしたバックアップにミラーがフェイルオーバーされるようにして、ミラーのアップグレードを完了します。

   ```
   icm upgrade -machine ".*(1|3)" -image intersystems/iris:2022.1.0.223.0
   ```

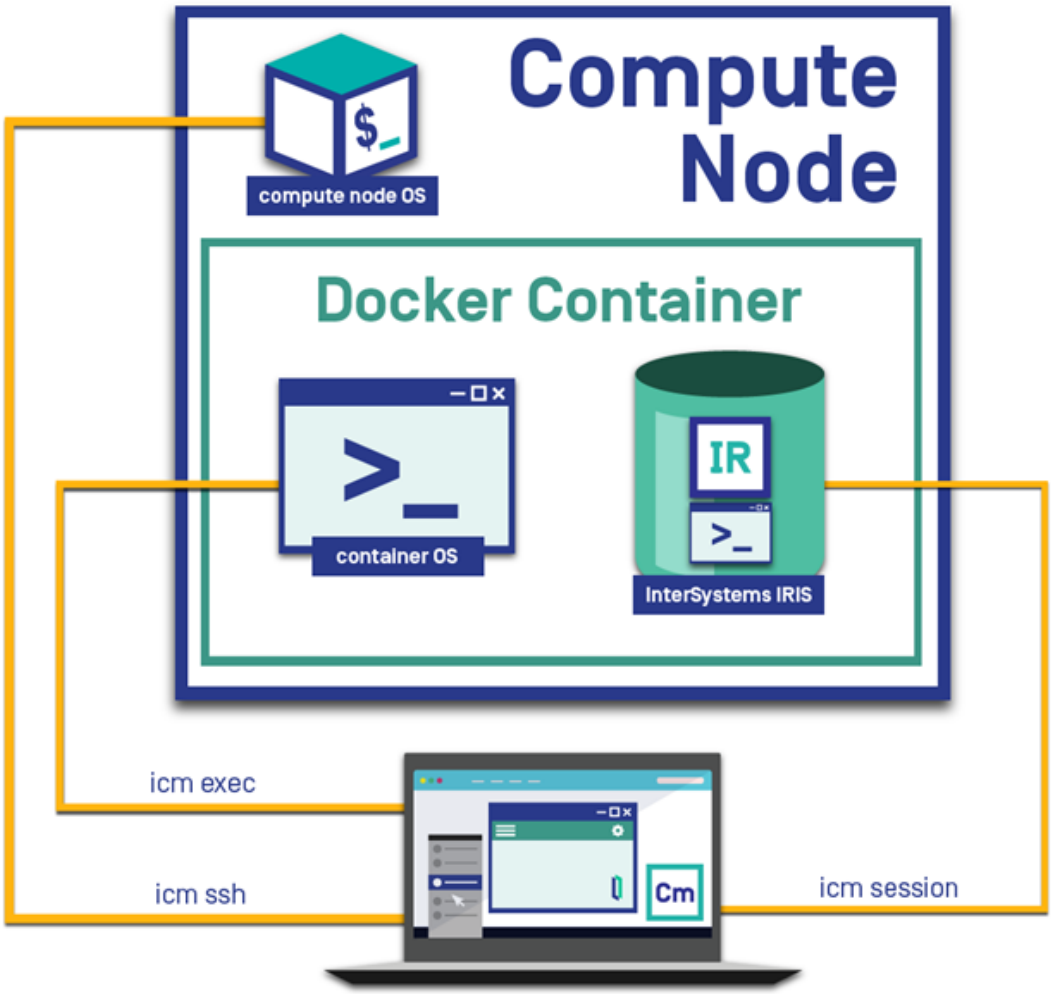
3.6.4 サービス管理コマンド

これらのコマンドを使用して、導入されたコンテナ内で実行されている InterSystems IRIS などのサービスとやり取りできます。

多くの ICM コマンド・オプションは、複数のコマンドで使用できます。例えば、–role オプションをさまざまなコマンドで使用して、コマンドを実行するノードのタイプを指定できます。例えば、icm exec –role AM では、導入環境にあるノードのうち、タイプが AM であるもののみで指定のコマンドを実行します。また、コンテナの導入元のイメージを指定する –image オプションは、icm run コマンドと icm upgrade コマンドの両方で使用できます。ICM コマンドとオプションの完全なリストについては、“ICM リファレンス” の章にある ″ICM コマンドとオプション″ を参照してください。

ICM の重要な機能は、導入環境のノードと複数のレベルでやり取りできることです（ノード自体、ノードに導入されたコンテナ、およびコンテナ内で実行中の InterSystems IRIS インスタンス）。icm ssh（″インフラストラクチャ管理コマンド″ を参照）を使用して、指定されたホスト・ノード上でコマンドを実行できます。このコマンドを、このセクションで先に説明した 2 つのコマンド icm exec（指定されたコンテナ内でコマンドを実行する）および icm session（指定されたノード上で InterSystems IRIS インスタンスのインタラクティブ・セッションを開く）と組み合わせて、ICM の導入を操作するための強力なツールのセットとして使用できます。以下の図に、この複数レベルでのやり取りを示します。

図 3-3: インタラクティブ ICM コマンド



3.6.4.1 icm exec

icm exec コマンドは、指定されたコンテナ内で任意のコマンドを実行します。例を以下に示します。

```
$ icm exec -command "df -k" -machine ANDY-DM-TEST-0001
Executing command in container iris on ANDY-DM-TEST-0001
...output in ./state/ANDY-DM-TEST/ANDY-DM-TEST-0001/docker.out

Filesystem     1K-blocks    Used Available Use% Mounted on
rootfs          10474496 2205468   8269028  22% /
tmpfs            3874116       0   3874116   0% /dev
tmpfs            3874116       0   3874116   0% /sys/fs/cgroup
/dev/xvda2      33542124 3766604  29775520  12% /host
/dev/xvdb       10190100   36888   9612540   1% /irissys/data
/dev/xvdc       10190100   36888   9612540   1% /irissys/wij
/dev/xvdd       10190100   36888   9612540   1% /irissys/journal1
/dev/xvde       10190100   36888   9612540   1% /irissys/journal2
shm                65536     492     65044   1% /dev/shm
```

複数のコマンドからの出力を混合すると解釈が難しくなるため、複数のノード上でコマンドが実行される場合、出力はファイルに書き込まれ、出力ファイルのリストが提供されます。

--env など、その他の Docker オプションは、-options オプションを使用して icm exec コマンド行に指定できます。詳細は、"カスタム・コンテナおよびサードパーティ・コンテナでの ICM の使用" を参照してください。

長時間実行されるコマンド、ブロック・コマンド、またはインタラクティブ・コマンドをコンテナ内で実行することで、ICM がコマンドの完了またはユーザ入力の待機中にタイムアウトする場合があるため、icm exec コマンドをインタラクティブ・モードで使用することもできます。単一ノードの導入環境でコマンドを実行する場合を除き、-interactive フラグと一緒に、コマンドを単一のノードに限定する -role オプションまたは -machine オプションを指定する必要があります。分かりやすい例として、コンテナ内でシェルを実行する場合を以下に示します。

```
$ icm exec -command bash -machine ANDY-AM-TEST-0004 -interactive
Executing command 'bash' in container iris on ANDY-AM-TEST-0004...
[root@localhost /] $ whoami
root
[root@localhost /] $ hostname
iris-ANDY-AM-TEST-0004
[root@localhost /] $ exit
```

コンテナ内でインタラクティブに実行されるコマンドのもう 1 つの例が、iris stop などの、ユーザ入力を求める InterSystems IRIS コマンドです。このコマンドは、InterSystems IRIS インスタンスをシャットダウンする前に、メッセージを送信するかどうかを尋ねます。

icm cp コマンドは、指定されたノード上のローカル・ファイルまたはディレクトリを指定されたコンテナにコピーするもので、icm exec で役立ちます。

3.6.4.2 icm session

-interactive オプションと共に使用すると、icm session コマンドは、指定されたノード上で InterSystems IRIS インスタンスのインタラクティブ・セッションを開きます。-namespace オプションを使用すると、セッションを開始するネームスペースを指定できます。既定値は、ICM によって作成されるネームスペース (既定では IRISCLUSTER) です。以下に例を示します。

```
$ icm session -interactive -machine ANDY-AM-TEST-0003 -namespace %SYS

Node: iris-ANDY-AM-TEST-0003, Instance: IRIS

%SYS>
```

-command オプションを使用して、InterSystems IRIS セッションで実行するルーチンを指定することもできます。以下に例を示します。

```
icm session -interactive -machine ANDY-AM-TEST-0003 -namespace %SYS -command ^MIRROR
```

--env など、その他の Docker オプションは、-options オプションを使用して icm exec コマンド行に指定できます。詳細は、"カスタム・コンテナおよびサードパーティ・コンテナでの ICM の使用" を参照してください。

-interactive オプションを使用しない場合、icm session コマンドは、指定されたノード上で、-command オプションで指定された InterSystems IRIS ObjectScriptScript スニペットを実行します。-namespace オプションを使用して、スニペットを実行するネームスペースを指定できます。複数のコマンドからの出力を混合すると解釈が難しくなるため、複数のノード上でコマンドが実行される場合、出力はファイルに書き込まれ、出力ファイルのリストが提供されます。以下に例を示します。

```
$ icm session -command 'Write ##class(%File).Exists("test.txt")' -role AM
Executing command in container iris on ANDY-AM-TEST-0003...
Executing command in container iris on ANDY-AM-TEST-0004...
Executing command in container iris on ANDY-AM-TEST-0005...
...output in ./state/ANDY-AM-TEST/ANDY-AM-TEST-0003/ssh.out
...output in ./state/ANDY-AM-TEST/ANDY-AM-TEST-0004/ssh.out
...output in ./state/ANDY-AM-TEST/ANDY-AM-TEST-0005/ssh.out
```

指定された -machine オプションまたは -role オプションによってコマンドが 1 つのノードに限定される場合、出力はコンソールにも書き込まれます。以下に例を示します。

```
$ icm session -command 'Write ##class(%File).Exists("test.txt")' -role DM
Executing command in container iris on ANDY-DM-TEST-0001
...output in ./state/ANDY-DM-TEST/ANDY-DM-TEST-0001/docker.out

0
```

icm sql コマンドは、指定されたノード (またはすべてのノード) 上のコンテナ化された InterSystems IRIS インスタンスに対して任意の SQL コマンドを実行するもので、icm session と似ています。

3.6.4.3 icm cp

icm cp コマンドは、指定されたノード上のローカル・ファイル・システムから指定されたコンテナにファイルまたはディレクトリをコピーします。また、-options --retrieve が指定されている場合は、コンテナからローカル・ファイル・システムにファイルまたはディレクトリをコピーします。

既定では、このコンテナは **iris** コンテナで、コピーはすべてのノードで実行されます。-container オプションを使用して別のコンテナを指定できます。また、-role オプションまたは -machine オプションを使用して、コピーの実行場所とするノードを指定できます。例えば、分散キャッシュ・クラスタにあるデータ・サーバ上でのみコピーを実行するには -role DM を指定します。**iris** コンテナではなく、導入したカスタム・コンテナとの間でコピーを実行するには -container container-name を指定します。

コマンドの基本的な構文は以下のとおりです。

```
icm cp -localPath local-path [-remotePath remote-path] [-options --retrieve]
```

-options --retrieve を指定すると、remote-path (コンテナでの絶対パス) から local-path (ローカル・ファイル・システム上の絶対パス) にファイルやディレクトリがコピーされます。省略すると、local-path から remote-path にコピーされます。localPath と remotePath の両方とも、ファイルまたはディレクトリのどちらも指定できます。両方がディレクトリの場合は、コピー元ディレクトリの内容が再帰的にコピーされます。ディレクトリ自体をコピーする場合は、それをコピー先のパスで指定します。remotePath 引数はオプションで、省略した場合は既定値の /tmp に設定されます。remotePath がディレクトリの場合は、末尾にスラッシュ (/) を付ける必要があり、そうでなければファイルと見なされます。

注釈　icm scp コマンドも参照してください。このコマンドは、ローカル ICM コンテナから指定ホスト OS にファイルまたはディレクトリを安全にコピーします。

ICM コマンド行に Docker 引数を指定できるようにする -options オプションの詳細は、"カスタム・コンテナおよびサードパーティ・コンテナでの ICM の使用" を参照してください。

3.6.4.4 icm sql

icm sql コマンドは、指定されたノード (またはすべてのノード) 上のコンテナ化された InterSystems IRIS インスタンスに対して、任意の SQL コマンドを実行します。例を以下に示します。

```
$ icm sql -command "SELECT Name,SMSGateway FROM %SYS.PhoneProviders" -role DM
Executing command in container iris on ANDY-DM-TEST-0001...
...output in ./state/ANDY-DM-TEST/ANDY-DM-TEST-0001/jdbc.out

Name,SMSGateway
AT&amp;T Wireless,txt.att.net
Alltel,message.alltel.com
Cellular One,mobile.celloneusa.com
Nextel,messaging.nextel.com
Sprint PCS,messaging.sprintpcs.com
T-Mobile,tmomail.net
Verizon,vtext.com
```

-namespace オプションを使用すると、SQL コマンドを実行するネームスペースを指定できます。既定値は、ICM によって作成されるネームスペース (既定では IRISCLUSTER) です。

複数のコマンドからの出力を混合すると解釈が難しくなるため、複数のノード上でコマンドが実行される場合、出力はファイルに書き込まれ、出力ファイルのリストが提供されます。

icm sql コマンドを単一のノード上でインタラクティブに実行して、InterSystems IRIS SQL シェルを開くこともできます (″InterSystems SQL の使用法″ の “SQL シェル・インタフェースの使用法” の章を参照してください)。単一ノードの導入環境でコマンドを実行する場合を除き、-interactive フラグと一緒に、コマンドを単一のノードに限定する -role オプションまたは -machine オプションを指定する必要があります。次に、例を示します。

```
$ icm sql -interactive -machine ANDY-QS-TEST-0002
SQL Command Line Shell
----------------------------------------------------
The command prefix is currently set to: <<nothing>>.
Enter <command>, 'q' to quit, '?' for help.
```

非インタラクティブ・コマンドと同様に、-namespace オプションをインタラクティブに使用して、SQL シェルを実行するネームスペースを指定できます。既定値は、ICM によって作成されるネームスペース (既定では IRISCLUSTER) です。

LB ノード (ロード・バランサ) のターゲット・プールが InterSystems IRIS ノード (DATA、COMPUTE、DM、または AM) で構成されている場合、-role オプションを使用して、SQL 呼び出しをその LB ノードに転送できます。以下に例を示します。

```
$ icm sql -role LB -command "SELECT * FROM TABLE1"
```

3.6.4.5 icm docker

icm docker コマンドは、指定されたノード (またはすべてのノード) 上で Docker コマンドを実行します。例を以下に示します。

```
$ icm docker -command "status --no-stream" -machine ANDY-DM-TEST-0002
Executing command 'status --no-stream' on ANDY-DM-TEST-0002...
...output in ./state/ANDY-DM-TEST/ANDY-DM-TEST-0002/docker.out

CONTAINER     CPU %  MEM USAGE / LIMIT  MEM %  NET I/O     BLOCK I/O        PIDS
3e94c3b20340  0.01%  606.9MiB/7.389GiB  8.02%  5.6B/3.5kB  464.5MB/21.79MB  0
1952342e3b6b  0.10%  22.17MiB/7.389GiB  0.29%  0B/0B       13.72MB/0B       0
d3bb3f9a756c  0.10%  40.54MiB/7.389GiB  0.54%  0B/0B       38.43MB/0B       0
46b263cb3799  0.14%  56.61MiB/7.389GiB  0.75%  0B/0B       19.32MB/231.9kB  0
```

Docker コマンドは長時間実行 (またはブロック) するものであってはなりません。これに従わなければ、ICM に制御が返されません。例えば、例の中で ---no-stream オプションを削除すると、タイムアウト期限が切れるまで呼び出しは戻りません。

3.7 インフラストラクチャのプロビジョニング解除

パブリック・クラウド・プラットフォームのインスタンスは継続的に課金され、プライベート・クラウド内で使用されていないインスタンスはリソースを無駄に消費するので、適切な時期にインフラストラクチャのプロビジョニングを解除することは重要です。

icm unprovision コマンドは、プロビジョニング時に作成された状態ファイルに基づいて、プロビジョニングされたインフラストラクチャの割り振りを解除します。**state** サブディレクトリが現在の作業ディレクトリ内にない場合は、その場所を指定するために -stateDir オプションが必要です。"インフラストラクチャのプロビジョニング" で説明されているように、**destroy** は、インフラストラクチャの割り振りを解除する Terraform のフェーズを指します。そのタイプのノードがいくつプロビジョニングされたかに関係なく、定義ファイル内のエントリごとに 1 行が作成されます。ICM は複数のスレッドで Terraform を実行するので、マシンのプロビジョニングが解除される順序は不定です。

```
$ icm unprovision -cleanUp
Type "yes" to confirm: yes
Starting destroy of ANDY-DM-TEST...
Starting destroy of ANDY-AM-TEST...
Starting destroy of ANDY-AR-TEST...
...completed destroy of ANDY-AR-TEST
...completed destroy of ANDY-AM-TEST
...completed destroy of ANDY-DM-TEST
Starting destroy of ANDY-TEST...
...completed destroy of ANDY-TEST
```

-cleanUp オプションは、プロビジョニング解除の後に状態ディレクトリを削除します。既定では、状態ディレクトリは保存されます。既定では、icm unprovision コマンドは、プロビジョニング解除を確認するプロンプトを出します。-force オプションを使用して、スクリプトの使用時などにこのプロンプトを回避できます。

4

ICM リファレンス

重要　　InterSystems IRIS リリース 2023.3 以降、InterSystems Cloud Manager (ICM) は非推奨になっています。今後のバージョンからは削除される予定です。

この章では、ICM のさまざまな要素とその使用法について詳しく説明します。

4.1 ICM コマンドとオプション

以下の最初の表は、ICM コマンド行で実行可能なコマンドの一覧です。それぞれのコマンドについては、“ICM の使用” の章で詳細に説明しています。

2 番目の表は、コマンドで使用可能なオプションの一覧です。コマンド行オプションには、以下の 2 つの目的があります。

- 必須またはオプションの引数をコマンドに指定する。例えば、導入環境にある InterSystems IRIS コンテナのみの実行状態をリストするには、次のコマンドを使用できます。

  ```
  icm ps -container iris
  ```

  ノード ANDY-DM-TEST 上に導入したコンテナ内部でシェルを開くコマンドを実行するには、次のコマンドを使用できます。

  ```
  icm exec -command bash -machine ANDY-DM-TEST -interactive
  ```

- フィールドの既定値または構成ファイルの値を上書きするには、以下の 2 つの方法のいずれかを使用します。
  - icm provision などのあらゆるコマンドで、-image、-namespace、および -iscpassword の各オプションを使用して、それぞれ **DockerImage**、**Namespace**、および **ISCPassword** の各フィールドの値を上書きできます。
  - プロビジョニング・フェーズの後、-overrides オプションを使用して、現在のコマンドでのみ 1 つ以上の値を上書きできます。例えば、既定値ファイルに以下のフィールドがあるとします。

    ```
    "DockerUsername": "prodriguez",
    "DockerPassword": "xxxxxxx",
    "DockerRegistry": "https://containers.intersystems.com",
    "DockerImage": "containers.intersystems.com/intersystems/iris:2022.1.0.223.0",
    ```

    -image オプションを指定して icm provision コマンドを実行すると **DockerImage** フィールドを上書きできますが、別のレジストリにあるイメージは使用できません。レジストリの場所と資格情報は上書きできないからです。ただ

し、icm upgrade コマンドでは、-overrides オプションを使用すると別のレジストリにあるイメージを指定して、以下のように 3 つのフィールドすべてを上書きできます。

```
icm upgrade -overrides '{"DockerUsername":"mwyszynska","DockerPassword":"xxxxxx",
  "DockerRegistry":"docker.io"}' -image docker.io/acme/iris:latest-em
```

注釈　-overrides を icm run、icm install、または icm upgrade の各コマンドで使用して、永続化を意図したフィールドを指定する場合、そのフィールドを instances.json ファイルの中で更新して、以降の再プロビジョニング・オペレーションで元に戻ることがないようにする必要があります。前述の icm upgrade コマンドに続いて、DockerImage、DockerRegistry、DockerUsername、DockerPassword などのフィールドをインスタンス・ファイルの中で更新する必要があります (-image、-namespace、および -iscpassword の各オプションでは、この処理が自動的に実行されます)。

両方の表に、関連したテキストへのリンクが示してあります。

注釈　コマンドの表は各コマンドで使用できるすべてのオプションを網羅しておらず、オプションの表は各オプションを指定できるすべてのコマンドを網羅していません。

テーブル 4-1: ICM コマンド

コマンド	説明	重要なオプション
provision	ホスト・ノードをプロビジョニングします	N/A
inventory	プロビジョニングされたホスト・ノードをリストします	-machine、-role、-json、-options
unprovision	ホスト・ノードを破棄します	-stateDir、-cleanup、-force
merge	別個のリージョンまたはプロバイダ・プラットフォームにプロビジョニングされたインフラストラクチャをマルチリージョンまたはマルチプロバイダ導入環境用の新しい定義ファイルにマージします	-options、-localPath
ssh	1 つ以上のホスト・ノード上でオペレーティング・システム・コマンドを実行します	-command、-machine、-role
scp	1 つ以上のホスト・ノードにローカル・ファイルをコピーします	-localPath、-remotePath、-machine、-role
run	ホスト・ノードにコンテナを導入します	-image、-container、-namespace、-options、-iscPassword、-command、-machine、-role、-override
ps	ホスト・ノードに導入されたコンテナの実行状態を表示します	-container、-json
stop	1 つ以上のホスト・ノード上でコンテナを停止します	-container、-machine、-role
start	1 つ以上のホスト・ノード上でコンテナを起動します	-container、-machine、-role
pull	1 つ以上のホスト・ノードにイメージをダウンロードします	-image、-container、-machine、-role

コマンド	説明	重要なオプション
rm	1 つ以上のホスト・ノードからコンテナを削除します	–container、–machine、–role
upgrade	1 つ以上のホスト・ノード上でコンテナを置き換えます	–image、–container、–machine、–role、–override
exec	1 つ以上のコンテナ内でオペレーティング・システム・コマンドを実行します	–container、–command、–interactive、–options、–machine、–role
session	コンテナ内の InterSystems IRIS インスタンスのインタラクティブ・セッションを開くか、1 つ以上のインスタンスで InterSystems IRIS ObjectScriptScript スニペットを実行します	–namespace、–command、–interactive、–options、–machine、–role
cp	1 つ以上のコンテナにローカル・ファイルをコピーします	–localPath、–remotePath、–machine、–role
sql	InterSystems IRIS インスタンス上で SQL 文を実行します	–namespace、–command、–machine、–role
install	コンテナレス・モードでキットから InterSystems IRIS インスタンスをインストールします	–machine、–role、–override
uninstall	コンテナレス・モードでキットからインストールされた InterSystems IRIS インスタンスをアンインストールします	–machine、–role
docker	1 つ以上のホスト・ノード上で Docker コマンドを実行します	–container、–machine、–role

テーブル 4–2: ICM コマンド行オプション

オプション	説明	既定	説明
–help	コマンドの使用法と ICM バージョンを表示します		–––
–version	ICM バージョンを表示します		–––
–verbose	実行の詳細を表示します	false	(すべてのコマンドに使用できます)
–force	プロビジョニング解除の前に確認を行いません	false	インフラストラクチャのプロビジョニング解除
–cleanUp	プロビジョニング解除の後に state ディレクトリを削除します	false	インフラストラクチャのプロビジョニング解除
–machine regexp	コマンドを実行するノードを指定するために使用するマシン名のパターン・マッチ	(すべて)	icm inventory、icm ssh、icm run、icm exec、icm session

オプション	説明	既定	説明
–role role	コマンドを実行する InterSystems IRIS インスタンスのロール (DATA や AM など)	(すべて)	icm inventory、icm ssh、icm run、icm exec、icm session
–namespace namespace	導入された InterSystems IRIS インスタンス上で作成し、session コマンドと sql コマンドの既定の実行ネームスペースとして設定するネームスペース	IRISCLUSTER	定義ファイル、icm run、icm session、icm sql
–image image	導入する Docker イメージ。リポジトリ名を記述している必要があります。	DockerImage 定義ファイルの値	icm run、icm upgrade
–override '{"field":"value",...}	このコマンドで上書きするフィールド値。	なし	ICM コマンドとオプション
–options options	コマンドで指定する Docker オプション。	なし	icm inventory、icm run、icm exec、icm session、複数のリージョンまたはプロバイダにわたる導入、カスタム・コンテナおよびサードパーティ・コンテナでの ICM の使用
–container name	コンテナの名前	icm ps コマンド : (すべて) その他のコマンド : iris	icm run、icm ps
–command cmd	実行するコマンドまたはクエリ	なし	icm ssh、icm run、icm exec、icm session、icm sql
–interactive	exec コマンドと ssh コマンドの入力/出力をコンソールにリダイレクトします	false	icm ssh、icm exec、icm sql
–localPath path	ノードのローカル・ファイル・システム上 (icm cp) または ICM コンテナ内部 (icm scp) のファイル・パスまたはディレクトリ・パス	なし	icm cp、icm scp、コンテナレスの導入、リモート・スクリプト呼び出し
–remotePath path	コンテナ内部 (icm cp) またはノードのローカル・ファイル・システム上 (icm scp) のファイル・パスまたはディレクトリ・パス	/home/SSHUser (SSHUser フィールドの値)	icm cp、icm scp、コンテナレスの導入、リモート・スクリプト呼び出し
–iscPassword password	導入される InterSystems IRIS インスタンスのパスワード	iscPassword 構成ファイルの値	icm run
–json	JSON 応答モードを有効にします	false	JSON モードの使用

重要　デバッグのみを目的としている -verbose オプションを使用すると、iscPassword の値やその他の機密情報 (DockerPassword など) が開示される場合があります。このオプションを使用する場合は、-force オプションも使用するか、続行する前に詳細モードを使用することを確認する必要があります。

4.2 ICM の構成パラメータ

以下の各テーブルでは、プロビジョニングと導入のタスク、および管理コマンドの実行に必要な情報を ICM に提供するために構成ファイルに組み込むことができるフィールドについて説明します (“基本的な ICM の要素” の章にある ″構成ファイル、状態ファイル、およびログ・ファイル″ および “ICM の使用” の章にある ″導入の定義″ を参照)。パラメータを名前で探すには、アルファベット順のリストを使用します。このリストには、パラメータの定義が記載されているテーブルへのリンクが用意されています。

- 一般パラメータ
- セキュリティ関連のパラメータ
- ポートおよびプロトコルのパラメータ
- CPF パラメータ
- プロバイダ固有のパラメータ
- デバイス名パラメータ
- ユーザ・パラメータのアルファベット順のリスト

4.2.1 一般パラメータ

以下のテーブルのフィールドはすべて、あらゆるクラウド・プロバイダで使用されます。一部のフィールドは、vSphere と PreExisting でも使用されます。

右端の 2 つの列は、各パラメータがすべての導入環境で必須であるか、オプションであるか、および defaults.json と definitions.json のどちらで指定する必要があるか (使用する場合)、どちらで推奨されるか、どちらでも使用できるかを示します。以下はその例です。

- 単一の導入環境は常に、(後で別のものとマージしてマルチプロバイダ導入環境を作成する場合でも) 選択した単一のプロビジョニング・プラットフォームを対象とするため、Provider パラメータは必須であり、既定値ファイルで指定する必要があります。
- それぞれのノード・タイプを指定する必要がありますが、1 つの導入環境に複数のノード・タイプを含めることができるため、Role パラメータは、定義ファイル内のそれぞれの定義で必須です。
- InterSystems IRIS を実行する各ノードにはライセンスが必要ですが、他のノードはライセンスを必要としないので、LicenseKey 設定は必須であり、通常は定義ファイル内の該当する定義で指定します。
- 少なくとも 1 つのコンテナを導入環境内の各ノードに導入する必要がありますが、単一のコンテナをすべてのノードに導入することもあれば (DATA ノードのみで構成されるシャード・クラスタ全体に **iris**/**iris-arm64** を導入する場合など)、それぞれのノード・タイプに異なるコンテナを導入することもあります (分散キャッシュ・クラスタ内の DM と AM には **iris**/**iris-arm64** を、WS には **webgateway** を、AR には **arbiter** を導入する場合など)。そのため、DockerImage パラメータは必須であり、既定値ファイル、定義ファイル、またはその両方で指定できます (両方で指定する場合は、既定のイメージを指定しますが、1 つ以上のノード・タイプについてそのイメージをオーバーライドします)。

- 導入するイメージと同様に、OS ボリュームのサイズも、既定値ファイルですべてのノードについて指定することも、定義ファイルで 1 つ以上のノード・タイプについて指定することも、両方で指定することもできます。ただし、既定値があるので、このパラメータはオプションです。

注釈　パラメータの既定値が記載されていない場合、既定値はありません。

パラメータ	定義	使用	構成ファイル
Provider	インフラストラクチャをプロビジョニングするプラットフォーム。"プロビジョニング・プラットフォーム" を参照してください。	必須	既定値ファイル
Label Tag	プロビジョニングされるクラウド・ノードの名前付け方式のフィールド：Label-Role-Tag-NNNN (例：ANDY-DATA-TEST-0001)。他のノードとの競合を避けるために、所有権と目的を示す必要があります。複数の導入環境で同じ Label および Tag を共有しないでください。ダッシュを使用することはできません。	必須	既定値ファイル
LicenseDir	ICM コンテナ内でステージングされ、LicenseKey フィールド (次の項目を参照) によって個々に指定される InterSystems IRIS ライセンス・キーの場所。"ICM の InterSystems IRIS ライセンス" を参照してください。	必須	既定値ファイル
LicenseKey	プロビジョニングされる 1 つ以上の DATA、COMPUTE、DM、AM、DS、または QS ノード上の InterSystems IRIS インスタンスのライセンス・キー。LicenseDir フィールド (前の項目を参照) で指定される場所の ICM コンテナ内でステージングされます。DM ノードと AM ノードのみを含む構成では、標準ライセンスを使用できます。他のすべての構成 (すなわち、シャード・クラスタ) では、シャーディング対応ライセンスが必要です。	必須	定義ファイル (推奨)
Region (Azure の同等のパラメータ：Location)	インフラストラクチャがプロビジョニングされる、プロバイダの計算リソースの地理的地域。単一の構成を複数のリージョンに導入する方法については、"複数のリージョンまたはプロバイダにわたる導入" を参照してください。プロバイダのドキュメントを含む、プロバイダ固有の情報は以下のとおりです。 ・ AWS ― 例：us-west-1。"AWS のリージョンとアベイラビリティ・ゾーンについて" を参照してください。 ・ GCP ― 例：us-east1。"リージョンとゾーン" を参照してください。マルチリージョン導入のコンマ区切りリストを指定できます。 ・ Azure ― 例：Central US。"Azure の地域" を参照してください(Region の代わりに Location を使用します)。 ・ Tencent ― 例：na-siliconvalley (West US/Silicon Valley)。"地域とアベイラビリティーゾーン" を参照してください。	必須	既定値ファイル

パラメータ	定義	使用	構成ファイル
Zone	プロビジョニングされるノードを配置する、指定したリージョン（前の項目を参照）内のアベイラビリティ・ゾーン。単一の構成を複数のゾーンに導入する方法については、"複数ゾーンにわたる導入" を参照してください。プロバイダ固有の情報は以下のとおりです。 ・ AWS — 例 : リージョン us-west-1 内の us-west-1c。"Regions and Zones" を参照してください。 ・ GCP — 例 : リージョン us-east1 内の us-east1-b。"リージョンとゾーン" を参照してください。 ・ Azure — 例 : リージョン Central US 内の 1。"可用性ゾーンとは" を参照してください。 注釈 一部の Azure リージョンは、アベイラビリティ・ゾーンをサポートしていません。このようなリージョンに導入するには、ゾーンを空文字列に設定するか、すべて省略します。 ・ Tencent — 例 : リージョン na-siliconvalley 内の na-siliconvalley-1。"地域とアベイラビリティーゾーン" を参照してください。	必須	既定値ファイル
ZoneMap	複数のゾーンにわたって導入する場合 ("複数ゾーンにわたる導入" を参照)、どのノードをどのゾーンに導入するかを指定します。既定値 : 0、1、2、...255。	オプション	定義ファイル
Mirror	true の場合、DATA ノード、DM ノード、および DS ノード上の InterSystems IRIS インスタンスはミラーとして導入されます。"ミラーリングされた構成の要件" を参照してください。既定値 : false。	オプション	既定値ファイル
MirrorMap	ミラーリングされる DATA、DS、および DM ノードのミラー・メンバ・タイプを指定します。DR 非同期ミラー・メンバを導入することもできます。"ミラーリングの規則" を参照してください。既定値 : primary,backup。async という用語は 1 回以上追加できます。例 : primary,backup,async,async。	オプション	定義ファイル
ISCPassword	プロビジョニングされる 1 つ以上のノード上の InterSystems IRIS インスタンスで事前定義のユーザ・アカウントに対して設定されるパスワード。対応するコマンド行オプション : –iscPassword。パラメータとオプションの両方を省略した場合、ICM によってパスワードの入力が求められます。詳細は、"icm run コマンド" を参照してください。	オプション	既定値ファイル
Namespace	導入される InterSystems IRIS インスタンス上に作成されるネームスペース。このネームスペースは、icm session コマンドと icm sql コマンドの既定のネームスペースですが、コマンド行オプション –namespace で指定またはオーバーライドすることもできます。既定値 : IRISCLUSTER。	オプション	既定値ファイル

パラメータ	定義	使用	構成ファイル
DockerImage	icm run コマンドによる導入で使用される Docker イメージ。リポジトリ名を含める必要があります (Docker ドキュメントの "Repositories" を参照)。defaults.json ですべてのノードについて指定し、オプションで definitions.json で特定のノード定義についてオーバーライドすることができます。コマンド行オプション –image を使用して指定またはオーバーライドすることもできます。	必須	
DockerRegistry	DockerImage によって指定されるイメージを保管する Docker リポジトリのホスト・サーバの DNS 名 (Docker ドキュメントの "About Registry" を参照)。このパラメータが含まれていない場合、ICM は docker.com にある Docker のパブリック・レジストリを使用します。InterSystems Container Registry (ICR) の詳細は、"ICM の使用" の章の "ICM イメージのダウンロード" を参照してください。	必須	既定値ファイル
DockerUsername	DockerRegistry (前の項目を参照) によって指定されたレジストリ上にある、DockerImage (前の項目を参照) で指定された Docker リポジトリにログインするために DockerPassword (次の項目を参照) と共に使用するユーザ名。パブリック・リポジトリの場合は必要ありません。このパラメータが含まれておらず、DockerImage によって指定されるリポジトリがプライベートの場合、ログインは失敗します。	必須	既定値ファイル
DockerPassword	Docker レジストリにログインするために DockerUsername (前の項目を参照) と共に使用するパスワード。パブリック・リポジトリの場合は必要ありません。このフィールドが含まれておらず、DockerImage によって指定されるリポジトリがプライベートの場合、パスワードを入力するように求められます (入力内容はマスクされます)。($、|、(、) などの特殊文字がこのフィールドの値に含まれる場合は、2 つの ¥ 文字でエスケープする必要があります。例えば、パスワード abc$def は、abc¥¥$def として指定する必要があります。)	必須	既定値ファイル

パラメータ	定義	使用	構成ファイル
DockerVersion	プロビジョニングされるノードにインストールされる Docker のバージョン。一般的に、各 /Samples/.../defaults.json のバージョンはプラットフォームに適合しています。ただし、組織で別のバージョンの Docker を使用している場合は、そのバージョンをノードにインストールする必要があります。 重要　インターシステムズのコンテナ・イメージは Open Container Initiative (OCI) の仕様に準拠しており、Docker Enterprise Edition エンジンを使用して構築されます。このエンジンは、OCI 標準を全面的にサポートしており、これによってイメージは認定され、Docker Hub レジストリで公開することができます。 インターシステムズのイメージは、広く知られたコンテナ向け Ubuntu オペレーティング・システムを使用して構築およびテストされているため、オンプレミスとパブリック・クラウドの両方において、Linux ベースのオペレーティング・システム上の OCI に準拠するすべてのランタイム・エンジンでサポートされます。	オプション	既定値ファイル
DockerURL	サブスクリプションまたは試用に関連付けられている Docker Enterprise Edition リポジトリの URL。これを指定すると、プロビジョニングされたノードに Docker Community Edition ではなく Docker Enterprise Edition のインストールがトリガされます。Docker EE の詳細は、Docker ドキュメントの "Docker Enterprise" を参照してください。	オプション	既定値ファイル
DockerInit	false に設定した場合、Docker --init オプションは、既定と異なり、InterSystems IRIS コンテナ以外のすべてのコンテナに渡されません。既定値 : true (--init オプションは InterSystems IRIS コンテナに渡されません)。	オプション	既定値ファイル
Overlay	Docker オーバーレイ・ネットワーク・タイプを決定します。通常は "weave" ですが、開発、パフォーマンス、またはデバッグを目的とする場合、または既存のクラスタへの導入時には "host" に設定できます。既定値 : weave (既存のクラスタへの導入時には host)。詳細は、Docker ドキュメントの "Use overlay networks" および Weave ドキュメントの "How the Weave Net Docker Network Plugins Work" を参照してください。	オプション	既定値ファイル
DockerStorageDriver	Docker によって使用されるストレージ・ドライバを指定します (Docker ドキュメントの "Docker storage drivers" を参照)。指定できる値は、overlay2 (既定値) と btrfs です。overlay2 に設定した場合、FileSystem (次の項目を参照) は xfs に設定し、btrfs に設定した場合、FileSystem は btrfs に設定する必要があります。	オプション	既定値ファイル

パラメータ	定義	使用	構成ファイル
FileSystem	プロビジョニングされるノードの永続ボリュームに使用するファイル・システムのタイプ。有効な値は xfs と btrfs です。既定値 : xfs。DockerStorageDriver (前の項目を参照) を overlay2 に設定した場合、FileSystem は xfs に設定する必要があります。DockerStorageDriver が btrfs の場合、FileSystem は btrfs である必要があります。	オプション	既定値ファイル (推奨)
OSVolumeSize	導入環境内のノードの OS ボリュームのサイズ (GB 単位)。既定値 : 32。マシン・イメージやテンプレート、インスタンス・タイプ、または OS ボリューム・タイプ・パラメータを指定する、該当パラメータに固有の設定によって制限されたり、それらの設定を優先して無視されることがあります (″プロバイダ固有のパラメータ″ を参照)。	オプション	
DataVolumeSize WIJVolumeSize Journal1VolumeSize Journal2VolumeSize	**iris** コンテナ用に作成する、対応する永続ストレージ・ボリュームのサイズ (GB 単位)。例えば、DataVolumeSize ではデータ・ボリュームのサイズを指定します。既定値は 10 ですが、Tencent の導入については DataVolumeSize を 60 以上にする必要があります。該当するボリューム・タイプ・パラメータによって制限されることがあります (″プロバイダ固有のパラメータ″ を参照)。それぞれのボリュームには、対応するデバイス名パラメータ (DataDeviceName など。″デバイス名パラメータ″ を参照) とマウント・ポイント・パラメータ (DataMountPoint など。すぐ後の項目と ″ICM によってマウントされるストレージ・ボリューム″ を参照) もあります。	オプション	
DataMountPoint WIJMountPoint Journal1MountPoint Journal2MountPoint	対応する永続ボリュームがマウントされる、**iris** コンテナ内の場所。例えば、DataMountPoint ではデータ・ボリュームの場所を指定します。詳細は、″ICM によってマウントされるストレージ・ボリューム″ を参照してください。既定値 : /irissys/{ data | wij | journal1j | journal2j }。それぞれのボリュームには、対応するデバイス名パラメータ (DataDeviceName など。″デバイス名パラメータ″ を参照) とサイズ・パラメータ (DataVolumeSize など。前の項目を参照) もあります。	オプション	
Containerless	true の場合、コンテナレス・モードが有効になり、コンテナではなく、インストール・キットから InterSystems IRIS が導入されます。付録 ″コンテナレスの導入″ を参照してください。既定値 : false。	オプション	既定値ファイル
Role	定義ファイルで指定されたエントリによってプロビジョニングされるノードのロール (DM や DATA など)。″ICM ノード・タイプ″ を参照してください。	必須	定義ファイル
Count	定義ファイルのエントリからプロビジョニングするノードの数。既定値 : 1。	必須	定義ファイル
StartCount	定義ファイル内の特定のノード定義で番号付けを開始する値。例えば、DS ノードの定義に ″StartCount″: ″3″ が含まれている場合、最初にプロビジョニングされる DS ノードの名前は Label-DS-Tag-0003 です。	オプション	定義ファイル

パラメータ	定義	使用	構成ファイル
LoadBalancer	ノード・タイプ DATA、COMPUTE、AM、または WS の定義で true の場合、事前定義されたロード・バランサがプロバイダ AWS、GCP、Azure、および Tencent に自動的にプロビジョニングされます（"事前定義されたロード・バランサ" を参照）。ノード・タイプ CN または VM の定義で true の場合、他のパラメータが定義に含まれていれば、汎用ロード・バランサが追加されます（"汎用ロード・バランサ" を参照）。既定値 ： false。	オプション	定義ファイル
AlternativeServers	タイプ WS の定義のリモート・サーバ選択アルゴリズム（"ノード・タイプ ： Web サーバ" を参照）。有効な値は LoadBalancing と FailOver です。既定値 ： LoadBalancing。	オプション	定義ファイル
ApplicationPath	タイプ WS の定義について作成するアプリケーション・パス。末尾にスラッシュを付けないでください。	オプション	定義ファイル
IAMImage	InterSystems API Manager (IAM) イメージ。既定値なし。	オプション	定義ファイル
PostgresImage	Postgres イメージ（オプションの IAM コンポーネント）。既定値 ： postgres:11.6。	オプション	定義ファイル
PrometheusImage	Prometheus イメージ（System Alerting and Monitoring [SAM] コンポーネント）。既定値 ： prom/prometheus:v2.17.1。	オプション	定義ファイル
AlertmanagerImage	Alertmanager イメージ（SAM コンポーネント）。既定値 ： prom/alertmanager:v0.20.0。	オプション	定義ファイル
GrafanaImage	Grafana イメージ（SAM コンポーネント）。既定値 ： grafana/grafana:6.7.1。	オプション	定義ファイル
NginxImage	Nginx イメージ（SAM コンポーネント）。既定値 ： nginx:1.17.9-alpine。	オプション	定義ファイル
UserCPF	導入時に InterSystems IRIS インスタンスの CPF をカスタマイズするために使用する構成マージ・ファイル（"カスタマイズされた InterSystems IRIS 構成を使用した導入" を参照）。	オプション	
SystemMode	プロビジョニングされる 1 つ以上のノード上の InterSystems IRIS インスタンスの管理ポータルのタイトルに表示される文字列。一部の値（LIVE、TEST、FAILOVER、DEVELOPMENT）を指定すると、表示にさらに変更が生じます。既定値 ： ブランク。この設定は、[Startup]/SystemMode を構成マージ・ファイルに追加することによって指定することもできます（前の項目を参照）。	オプション	

4.2.2 セキュリティ関連のパラメータ

以下のテーブルのパラメータは、プロビジョニングされたノードおよび導入されたコンテナと ICM が安全に通信できるように、アクセスを提供し、必要なファイルと情報を指定するために使用されます。これらはすべて必須で、既定値ファイルのみで指定します。

- ICM で提供されているスクリプトを使用してこれらのファイルを生成する方法については、“ICM の使用” の章にある "セキュリティ関連ファイルの入手" を参照してください。

- プロビジョニングされたノードおよびそれらのノード上のサービスと安全に通信するために、提供されたセキュリティ・ファイルを ICM がどのように使用するかについては、この章の "ICM セキュリティ" を参照してください。
- SSH プロトコルの一般的な使用法については、SSH Communications Security の "SSH PROTOCOL" を参照してください。
- Docker での TLS 証明書の使用を含め、Docker セキュリティの詳細は、Docker ドキュメントの "Docker security" を参照してください。
- InterSystems IRIS での TLS の使用に関する一般情報は、インターシステムズの "TLS ガイド" と "インターシステムズ公開鍵インフラストラクチャ" を参照してください。SSLConfig パラメータによって指定されるファイルの内容については、"クライアント構成の生成" を参照してください。
- TLS を使用してミラー・メンバ間の接続を保護する方法については、"高可用性ガイド" の "TLS セキュリティを使用したミラー通信の保護" を参照してください。

パラメータ	定義
プロバイダ固有の資格情報およびアカウント・パラメータ (ファイルおよび値を入手するための詳細な手順を確認するには、プロバイダのリンクをクリックしてください)	• プロバイダ固有 – AWS Credentials : AWS アカウントの公開/秘密鍵ペアを含むファイルへのパス。 • プロバイダ固有 – GCP Credentials : GCP アカウントのサービス・アカウント・キーを含む JSON ファイルへのパス。 Project : GCP プロジェクト ID。 • プロバイダ固有 – Azure SubscriptionId : Microsoft Azure サブスクリプションを識別する一意の英数字文字列。 TenantId : アプリケーションが作成された Azure Active Directory ディレクトリを識別する一意の英数字文字列。 UseMSI : true の場合、ClientId と ClientSecret の代わりにマネージド・サービス ID を使用して認証します。既定値は false です。 ClientId、ClientSecret : Azure アプリケーションを識別し、アプリケーションへのアクセスを提供する資格情報 (UseMSI が false である場合)。 • プロバイダ固有 – Tencent SecretID、SecretKey : Tencent Cloud アカウントを識別し、アカウントへのアクセスを提供する一意の英数字文字列。 • プロバイダ固有 – vSphere VSphereUser、VSpherePassword : vSphere を操作するための資格情報。

パラメータ	定義
SSHUser	プロビジョニングされたノードにアクセスするために ICM によって使用される、sudo アクセス権を持つ非 root アカウント。SSHUser のホーム・ディレクトリのルートは、Home フィールドを使用して指定できます。必要な値は、以下のようにプロバイダ固有です。 ・ AWS — AMI に基づく (″Amazon Web Services (AWS Parameters)″ で AMI パラメータを参照)。通常、Ubuntu イメージの場合は ubuntu ・ GCP — ユーザの任意 ・ Azure — ユーザの任意 ・ Tencent — イメージに基づく (″Tencent Cloud (Tencent) パラメータ″ の ″ImageId″ パラメータを参照) ・ vSphere — VM テンプレートに基づく (″VMware vSphere (vSphere) パラメータ″ の ″Template″ パラメータを参照) ・ PreExisting — 付録 “既存のクラスタへの導入” の ″SSH″ を参照
SSHPassword	SSHUser によって指定されたユーザの初期パスワード。マーケットプレイスから入手した Docker イメージ、およびタイプ vSphere、Azure、および PreExisting の導入には必須。このパスワードはプロビジョニング時のみに使用され、その完了時にパスワード・ログインは無効になります。
SSHOnly	true の場合、ICM はプロビジョニング時に SSH パスワード・ログインを試行しません (プロバイダ vSphere および PreExisting の場合のみ)。これにより、ICM はパスワードを使用してログインできなくなるので、SSH 公開鍵 (次の SSHPublicKey フィールドによって指定される) を各ノードでステージングする必要があります。既定値 : false。

SSHPublicKey	SSH 公開鍵/秘密鍵ペアの公開鍵の ICM コンテナ内のパス。すべての導入環境について必須です。プロバイダ AWS については、SSH2 形式である必要があります。以下に例を示します。
	`---- BEGIN SSH2 PUBLIC KEY ---`
	`AAAAB3NzaC1yc2EAAAABJQAAAQEAoa0`
	`---- BEGIN SSH2 PUBLIC KEY ---`
	その他のプロバイダについては、OpenSSH 形式である必要があります。以下に例を示します。
	`ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAQEAoa0`

SSHPrivateKey	SSH 公開鍵/秘密鍵ペアの秘密鍵の ICM コンテナ内のパス。すべての導入環境について必須で、RSA 形式である必要があります。以下に例を示します。
	`-----BEGIN RSA PRIVATE KEY-----`
	`MIIEogIBAAKCAQEAoa0ex+JKzC2Nka1`
	`-----END RSA PRIVATE KEY-----`

パラメータ	定義
TLSKeyDir	Docker、インターシステムズの Web ゲートウェイ、JDBC、およびミラーリングされた InterSystems IRIS データベースへのセキュリティ保護された接続を確立するために使用される TLS 鍵を含む ICM コンテナ内のディレクトリ。以下に例を示します。 ・ ca.pem ・ cert.pem ・ key.pem ・ keycert.pem ・ server-cert.pem ・ server-key.pem ・ keystore.p12 ・ truststore.jks ・ SSLConfig.properties
SSLConfig	セキュリティ保護された JDBC 接続を確立するために使用される TLS 構成ファイルへの ICM コンテナ内のパス。既定 : このパラメータが指定されない場合、ICM は /TLSKeyDir/SSLConfig.Properties 内で構成ファイルを探します (前の項目を参照)。
PrivateSubnet	true の場合、ICM は既存のプライベート・サブネットで導入を行うか、要塞ホストで使用する新しいプライベート・サブネットを作成して導入を行います。"プライベート・ネットワークへの導入" を参照してください。
WeavePassword	Weave Net 上のトラフィックの暗号化に使用するパスワード。暗号化を有効にするには、既定値ファイルで null 以外の値を設定します。既定値 : null。
net_vpc_cidr	導入を行う既存のプライベート・ネットワークの CIDR。"既存のプライベート・ネットワーク内での導入" を参照してください。
net_subnet_cidr	既存のプライベート・ネットワーク内の ICM ノードのサブネットの CIDR。

4.2.3 ポートおよびプロトコルのパラメータ

通常、これらのパラメータについては既定値で十分です。これらのパラメータのいくつかを指定する必要が生じる可能性がある 2 つの使用事例については、付録 “カスタム・コンテナおよびサードパーティ・コンテナでの ICM の使用” の "ポート" および付録 “既存のクラスタへの導入” の "ポート" を参照してください。

パラメータ	定義

パラメータ	定義
ForwardPort	指定されたロード・バランサによって転送されるポート('転送元' と '転送先' の両方)。既定値 : ・ AM、DM、DATA、COMPUTE : SuperServerPort、WebServerPort (以下を参照) ・ WS : WebGatewayPort (以下を参照) ・ VM/CN : ユーザ定義 (汎用ロード・バランサを導入するには、含まれている必要があります) すべてのポートで同じ ForwardProtocol (以下を参照) を使用する限り、値にはポートのコンマ区切りリストを指定できます。
ForwardProtocol	指定されたロード・バランサによって転送されるプロトコル。 値 TCP はすべてのプロバイダに有効です。プロバイダごとに追加のプロトコルを使用できます。 ・ DATA、COMPUTE、DM、AM : TCP ・ WS : TCP ・ VM/CN : ユーザ定義 (汎用ロード・バランサを導入するには、パラメータが含まれている必要があります)
HealthCheckPort	ターゲット・プール内のインスタンスの正常性を検証するために使用されるポート。既定値 : ・ AM、DM、DATA、COMPUTE : WebServerPort (以下を参照) ・ WS : 80 ・ VM/CN : ユーザ定義 (汎用ロード・バランサを導入するには、パラメータが含まれている必要があります)
HealthCheckProtocol	ターゲット・プール内のインスタンスの正常性を検証するために使用されるプロトコル。既定値 : ・ AM、DM、DATA、COMPUTE : HTTP ・ WS : TCP ・ VM/CN : ユーザ定義 (汎用ロード・バランサを導入するには、パラメータが含まれている必要があります)
HealthCheckPath	ターゲット・プール内のインスタンスの正常性を検証するために使用されるパス。既定値 : ・ ミラーリングされていない DM/DATA、AM、COMPUTE : /csp/user/cache_status.cxw ・ ミラーリングされている DM、DATA : /csp/user/mirror_status.cxw ・ WS : 該当なし (TCP 正常性チェックにパスは使用されません) ・ VM/CN : HTTP 正常性チェック用にユーザが定義 (汎用ロード・バランサを導入するには、パラメータが含まれている必要があります)

パラメータ	定義
ISCAgentPort *	InterSystems IRIS ISC エージェントによって使用されるポート。既定値 : 2188。Containerless が false であるか、指定されておらず、Overlay が weave に設定されている場合 ("一般パラメータ" を参照)、このポートはファイアウォールで閉じられます。
SuperServerPort	InterSystems IRIS スーパーサーバによって使用されるポート。既定値 : 1972。
WebServerPort	InterSystems IRIS Web サーバ/管理ポータルによって使用されるポート。既定値 : 52773。非 root のコンテナレス・モードで WS ノードに導入されているインターシステムズの Web ゲートウェイ・インスタンスでも使用されます。
WebGatewayPort	InterSystems IRIS Web ゲートウェイによって使用されるポート。既定値 : 80 (**webgateway**、**webgateway-nginx**)、52773 (**webgateway-lockeddown**)。
LicenseServerPort *	InterSystems IRIS ライセンス・サーバによって使用されるポート。既定値 : 4002。Containerless が false であるか、指定されておらず、Overlay が weave に設定されている場合 ("一般パラメータ" を参照)、このポートはファイアウォールで閉じられます。

* ICM がコンテナ・モードであり (Containerless が false であるか、指定されていない)、Overlay が weave に設定されている場合 ("一般パラメータ" を参照)、このポートはノードのファイアウォールで閉じられています。

4.2.4 CPF パラメータ

"カスタマイズされた InterSystems IRIS 構成を使用した導入" の説明に従って、UserCPF プロパティによって指定された構成マージ・ファイルを使用して、導入時に 1 つ以上の InterSystems IRIS インスタンスの CPF をカスタマイズする場合、含めることができない CPF 設定があります。ICM は先にそれらの値を読み取る必要があり、そのうえで後から CPF に追加するためです。したがって、構成ファイルで以下のパラメータ ("一般パラメータ" および "ポートおよびプロトコルのパラメータ" を参照) を指定することによって、これらの設定をカスタマイズする必要があります。

パラメータ	CPF 設定
WIJMountPoint	[config]/wijdir
Journal1MountPoint	[Journal]/CurrentDirectory
Journal2MountPoint	[Journal]/AlternateDirectory
SuperServerPort	[Startup]/DefaultPort
WebServerPort	[Startup]/WebServerPort

注釈　ICM の LicenseServerPort フィールドの値は、CPF の [LicenseServers] ブロックから取得され、構成されているライセンス・サーバ ("ICM の InterSystems IRIS ライセンス" を参照) の名前にバインドされます。

4.2.5 プロバイダ固有のパラメータ

このセクションの表は、各種クラウド・プロバイダに固有な、ICM で使用されるパラメータの一覧です。これらのパラメータの中には、複数のプロバイダで使用されるものがあります。例えば、InstanceType、ElasticIP、および VPCId の各パラメータは AWS と Tencent の両方の導入で使用できます。プロバイダ固有のパラメータの中には、名前が違っても目的

が同じものがあります。例えば、AWS の **AMI** と **InstanceType**、GCP の **Image** と **MachineType**、および Tencent の **ImageId** と **InstanceType** です。一方、これらのそれぞれに対応する 4 つの Azure パラメータがあります。

"一般パラメータ" のテーブルと同様に、このセクションのテーブルには、各パラメータがすべての導入環境で必須であるか、オプションであるか、および **defaults.json** と **definitions.json** のどちらで指定する必要があるか (使用する場合)、どちらで推奨されるか、どちらでも使用できるかを示します。それぞれのタイプの例については、"一般パラメータ" を参照してください。

注釈　PreExisting の導入のみに使用されるパラメータについては、付録 "既存のクラスタへの導入" の "PreExisting の定義ファイル" を参照してください。

4.2.5.1 マシン・イメージの選択

クラウド・プロバイダは、世界のさまざまな地域でデータ・センタを運営しています。このため、導入環境に合わせてカスタマイズする重要な項目の 1 つは、クラスタが導入される地域です ("一般パラメータ" の "Region" パラメータを参照してください)。もう 1 つの選択は、クラスタ内でホスト・ノードにどの仮想マシン・イメージを使用するかということです (パラメータはプロバイダによって異なります)。サンプル構成ファイルにはすべてのクラウド・プロバイダに有効な地域とマシン・イメージが定義されていますが、通常は個々のユーザの場所に合うように地域を変更する必要があります。マシン・イメージは地域に固有である場合が多いため、両方を選択する必要があります。

インターシステムズのコンテナ・イメージは Open Container Initiative (OCI) の仕様に準拠しており、Docker Enterprise Edition エンジンを使用して構築されます。このエンジンは、OCI 標準を全面的にサポートしており、これによってイメージは認定され、Docker Hub レジストリで公開することができます。インターシステムズのイメージは、広く知られたコンテナ向け Ubuntu オペレーティング・システムと ICM を使用して構築およびテストされているため、オンプレミスとパブリック・クラウドの両方において、Linux ベースのオペレーティング・システム上の OCI に準拠するすべてのランタイム・エンジンで導入がサポートされます。

4.2.5.2 プロバイダ固有のパラメータの表

テーブル 4-3: AWS パラメータ

パラメータ	定義	使用	構成ファイル
Credentials	AWS アカウントの公開鍵/秘密鍵ペアを含むファイルへのパス。ダウンロードするには、AWS 管理コンソールにログインした後、AWS ドキュメントの "アクセスキーの管理 (コンソール)" を開き、AWS コンソールでアクセス・キーを管理するための手順に従います。	必須	既定値ファイル
AMI	プロビジョニングされるノードのプラットフォームおよび OS のテンプレートとして使用する AMI (マシン・イメージ)。AWS ドキュメントの "Amazon マシンイメージ (AMI)" を参照してください。例 : ami-a540a5e1。使用可能なパブリック AMI をリストするには、EC2 コンソールのナビゲーション・ペインで [AMIs] を選択し、[Public AMIs] でフィルタリングします。	必須	
InstanceType	AWS および Tencent 上にプロビジョニングされるノードの計算リソースのテンプレートとして使用するインスタンス・タイプ。AWS ドキュメントの "Amazon EC2 インスタンスタイプ" を参照してください。例 : m4.large。(一部のインスタンス・タイプは、一部の AMI と互換性がない場合があります。)	必須	
ElasticIP	AWS および Tencent で Elastic IP 機能を有効にして、ホスト・ノードの再起動後も IP アドレスとドメイン名を保持します ("ホスト・ノードの再起動とリカバリ" を参照)。既定値 : false。	オプション	既定値ファイル

パラメータ	定義	使用	構成ファイル
VPCId	AWS および Tencent への導入に使用する既存の仮想プライベート・クラウド (VPC)。新しいものは割り当てません。指定した VPC が、プロビジョニング解除の際に割り当て解除されることはありません。指定しない場合、PrivateSubnet (″セキュリティ関連のパラメータ″ を参照) が true であれば、導入環境に新しい VPC が割り当てられ、プロビジョニング解除の際に割り当て解除されます。詳細は、″既存のプライベート・ネットワーク内での導入″ を参照してください。 注釈　既定のアドレス空間 10.0.%d.0/24 に VPC を作成しない場合は、内部パラメータ net_subnet_cidr を指定する必要があります。例えば、範囲が 172.17.0.0/24 の VPC の場合、net_subnet_cidr を 172.17.%d.0/24 として指定する必要があります。	オプション	既定値ファイル
SubnetIds	AWS または Tencent の既存のプライベート・サブネットで導入を行う場合、サブネット ID のコンマ区切りリスト。Zone パラメータ (″一般パラメータ″ を参照) によって指定された要素ごとに 1 つずつサブネット ID を追加します。	オプション	既定値ファイル
RouteTableId	既存のプライベート・サブネット上で導入する場合、ICM ホストへのアクセスに使用するルート・テーブル。指定すると、ICM では独自のテーブルを割り当てる代わりに、これを使用します (プロビジョニング解除の際に割り当て解除されません)。既定値なし。	オプション	既定値ファイル
InternetGatewayId	既存のプライベート・サブネットで導入を行う場合、ICM ホストへのアクセスに使用するインターネット・ゲートウェイ。指定されている場合、ICM は独自のものを割り当てる代わりに、これを使用します (プロビジョニング解除の際に割り当て解除しません)。既定値なし。	オプション	既定値ファイル
OSVolumeType	導入環境内のノードの OS ボリュームのディスク・タイプを指定します。これにより、OS ボリュームのサイズを設定する OSVolumeSize パラメータ (″一般パラメータ″ を参照) の最大値が決まります。AWS ドキュメントの ″Amazon EBS ボリュームの種類″ を参照してください。Tencent でも同じパラメータ名が使用されます。既定値 : standard。	オプション	
DataVolumeType WIJVolumeType Journal1VolumeType Journal2VolumeType	**iris** コンテナ用の対応する永続ストレージ・ボリューム (″ICM によってマウントされるストレージ・ボリューム″ を参照) のディスク・タイプを指定します。これにより、ボリュームの最大サイズが決まります。例えば、DataVolumeType によって、データ・ボリュームのサイズを指定する DataVolumeSize パラメータ (″一般パラメータ″ を参照) の最大値が決まります。AWS ドキュメントの ″Amazon EBS ボリュームの種類″ を参照してください。Tencent でも同じパラメータ名が使用されます。既定値 : standard。	オプション	
OSVolumeIOPS	導入環境内のノードの OS ボリュームの IOPS 数を指定します。AWS ドキュメントの ″I/O の特性とモニタリング″ を参照してください。既定値 : 0。	オプション	

パラメータ	定義	使用	構成ファイル
PlacementGroups	作成するプレイスメント・グループのコンマ区切りリスト (AWS ドキュメントの "プレイスメントグループ" を参照)。空白のままにするか、省略した場合、プレイスメント・グループは作成されません。既定値 : なし。	オプション	
PlacementStrategy	PlacementGroups で指定したグループへのインスタンスの配置方法。有効な値は、cluster、partition、spread です。既定値 : cluster。	オプション	
PlacementMap	PlacementGroups の値と、指定した定義内のノードとのマッピングを指定します。インスタンスは、PlacementGroups に記述された順序で割り当てられます (ラップアラウンドあり)。既定値 : 0、1、2、3、...256。	オプション	
PlacementPartitionCount	プレイスメント・グループに作成するパーティションの数。PlacementStrategy を partition に設定しないと効果が得られません。既定値 : 2。	オプション	
PlacementSpreadLevel	個別のハードウェアにインスタンスのグループを配置します。PlacementStrategy を spread に設定しないと効果が得られません。有効な値は rack と host です。既定値 : なし	オプション	
DataVolumeIOPS WIJVolumeIOPS Journal1VolumeIOPS Journal2VolumeIOPS	**iris** コンテナ用の対応する永続ストレージ・ボリューム ("ICM によってマウントされるストレージ・ボリューム" を参照) の IOPS 数を指定します。例えば、DataVolumeIOPS ではデータ・ボリュームの IOPS 数を指定します。AWS ドキュメントの "I/O の特性とモニタリング" を参照してください。対応するボリューム・タイプ (すぐ前の項目を参照) が io1 の場合は、0 以外にする必要があります。既定値 : 0。	オプション	
LoadBalancerInternal	True に設定すると "internal" タイプのロード・バランサが作成され、それ以外は "external" タイプのロード・バランサが作成されます。既定値 : False。	オプション	定義ファイル

テーブル 4-4: GCP パラメータ

パラメータ	定義	使用	構成ファイル

パラメータ	定義	使用	構成ファイル
Credentials	GCP アカウントのサービス・アカウント・キーを含む JSON ファイルへのパス。ダウンロードするには、GCP コンソールにログインし、プロジェクトを選択した後、GCP ドキュメントの "サービスアカウントキーの作成と管理" を開き、GCP コンソールでサービス・アカウント・キーを作成するための手順に従います。	必須	既定値ファイル
Project	GCP プロジェクト ID。GCP ドキュメントの "プロジェクトの作成と管理" を参照してください。	必須	既定値ファイル
Image	プロビジョニングされるノードのプラットフォームおよび OS のテンプレートとして使用するソース・マシン・イメージ。GCP ドキュメントの "イメージ" を参照してください。例 : ubuntu-os-cloud/ubuntu-1804-bionic-v20190911。	必須	
MachineType	プロビジョニングされるノードの計算リソースのテンプレートとして使用するマシン・タイプ。GCP ドキュメントの "マシンタイプ" を参照してください。例 : n1-standard-1。	必須	
RegionMap	複数のリージョンにわたって導入する場合 ("GCP での複数のリージョンにわたる導入" を参照)、どのノードをどのリージョンに導入するかを指定します。既定値 : 0、1、2、...255。	オプション	定義ファイル
Network	導入に使用する既存の仮想プライベート・クラウド (VPC)。新しいものは割り当てません。指定した VPC が、プロビジョニング解除の際に割り当て解除されることはありません。指定しない場合、PrivateSubnet ("セキュリティ関連のパラメータ" を参照) が true であれば、導入環境に新しい VPC が割り当てられ、プロビジョニング解除の際に割り当て解除されます。詳細は、"既存のプライベート・ネットワーク内での導入" を参照してください。	オプション	既定値ファイル
Subnet	導入環境で使用する既存のプライベート・サブネット。新しいものは割り当てません。プロビジョニング解除の際に割り当て解除されることはありません。マルチリージョン導入の場合 ("GCP での複数のリージョンにわたる導入" を参照)、値は指定されたリージョンごとに 1 つのコンマ区切りリストである必要があります。指定しない場合、PrivateSubnet ("セキュリティ関連のパラメータ" を参照) が true であれば、導入環境に新しい VPC が割り当てられ、プロビジョニング解除の際に割り当て解除されます。詳細は、"既存のプライベート・ネットワーク内での導入" を参照してください。	オプション	既定値ファイル
OSVolumeType	導入環境内のノードの OS ボリュームのディスク・タイプを指定します。GCP ドキュメントの "ストレージ オプション" を参照してください。既定値 : pd-standard。	オプション	
DockerVolumeType	導入環境内のノードの Docker シン・プールに使用するブロック・ストレージ・デバイスのディスク・タイプを指定します。GCP ドキュメントの "ストレージ オプション" を参照してください。既定値 : pd-standard。	オプション	

パラメータ	定義	使用	構成ファイル
DataVolumeType WIJVolumeType Journal1Volume-Type Journal2Volume-Type	**iris** コンテナ用の対応する永続ストレージ・ボリューム (″ICM によってマウントされるストレージ・ボリューム″ を参照) のディスク・タイプを指定します。例えば、DataVolumeType ではデータ・ボリュームのディスク・タイプを指定します。GCP ドキュメントの ″ストレージ オプション″ を参照してください。既定値 : pd-standard。	オプション	

テーブル 4-5: Azure パラメータ

パラメータ	定義	使用	構成ファイル
SubscriptionId	Microsoft Azure サブスクリプションを識別する一意の英数字文字列。表示するには、Azure ポータルで [サブスクリプション] を選択するか、検索ボックスに “サブスクリプション” と入力し、表示される [サブスクリプション ID] を SubscriptionId に使用します。	必須	既定値ファイル
TenantId	アプリケーションが作成された Azure Active Directory ディレクトリを識別する一意の英数字文字列。表示するには、Azure ポータルのナビゲーション・ペインで [Azure Active Directory] を選択した後、そのページのナビゲーション・ペインで [プロパティ] を選択し、表示される [ディレクトリ ID] を TenantId に使用します。	必須	既定値ファイル
UseMSI	true の場合、クライアント ID とクライアントの秘密鍵の代わりにマネージド・サービス ID を使用して認証します。Azure ドキュメントの ″Azure リソースのマネージド ID とは″ を参照してください。Azure にあるマシンから ICM を実行する必要があります。	必須	既定値ファイル
ClientId ClientSecret	Azure アプリケーションを識別し、そのアプリケーションへのアクセスを提供する資格情報 (UseMSI が false である場合)。作成する手順は、以下のとおりです。 · ″クイック スタート:Microsoft ID プラットフォームにアプリケーションを登録する″ に記載されている手順に従って、新しいアプリケーション登録を作成します。 · [アプリの登録] タブに表示される [アプリケーション ID] を ClientId に使用します。 · [設定]→[キー] を選択してキーを生成し、表示されるキー値を ClientSecret に使用します。	必須	既定値ファイル
Location	ノードをプロビジョニングするリージョン。″一般パラメータ″ の ″Region″ パラメータを参照してください。	必須	既定値ファイル
LocationMap	複数のリージョンにわたって導入する場合 (″Azure での複数のリージョンにわたる導入″ を参照)、どのノードをどのリージョンに導入するかを指定します。既定値 : 0、1、2、...255。	オプション	定義ファイル

パラメータ	定義	使用	構成ファイル
PublisherName	プロビジョニングされるノードのプラットフォームおよび OS のテンプレートとして使用する、指定された Azure マシン・イメージを提供するエンティティ。例 : OpenLogic。	必須	
Offer	指定された Azure マシン・イメージのオペレーティング・システム。例 : UbuntuServer。	必須	
Sku	指定された Azure マシン・イメージのオペレーティング・システムのメジャー・バージョン。例 : 7.2。	必須	
Version	指定された Azure マシン・イメージのビルド・バージョン。例 : 7.2.20170105。	必須	
CustomImage	PublisherName、Offer、Sku、および Version の各フィールドで表される Azure マシン・イメージの代わりに、OS ディスクの作成に使用されるイメージ。値は、以下の形式の Azure URI です。 /subscriptions/subscription/resource-Groups/resource_group/providers /Microsoft.Com-pute/images/image_name	オプション	
Size	プロビジョニングされるノードの計算リソースのテンプレートとして使用するマシン・サイズ。Azure ドキュメントの "Azure の仮想マシンのサイズ" を参照してください。例 : Standard_DS1。	必須	
ResourceGroupName	導入環境で使用する既存のリソース・グループ。新しいものは割り当てません。指定したグループが、プロビジョニング解除の際に割り当て解除されることはありません。指定しない場合、PrivateSubnet ("セキュリティ関連のパラメータ" を参照) が true であれば、導入環境に新しいリソース・グループが割り当てられ、プロビジョニング解除の際に割り当て解除されます。詳細は、"既存のプライベート・ネットワーク内での導入" を参照してください。	オプション	既定値ファイル
VirtualNetworkName	導入環境で使用する既存のプライベート・サブネット。新しいものは割り当てません。プロビジョニング解除の際に割り当て解除されることはありません。マルチリージョン導入環境の場合 ("Azure での複数のリージョンにわたる導入" を参照)、値は指定されたリージョンごとに 1 つのコンマ区切りリストである必要があります。指定しない場合、PrivateSubnet ("セキュリティ関連のパラメータ" を参照) が true であれば、導入環境に新しい VPC が割り当てられ、プロビジョニング解除の際に割り当て解除されます。詳細は、"既存のプライベート・ネットワーク内での導入" を参照してください。 注釈 既定のアドレス空間 10.0.%d.0/24 にネットワークを作成しない場合は、net_subnet_cidr パラメータ ("セキュリティ関連のパラメータ" を参照) を指定する必要があります。	オプション	既定値ファイル

パラメータ	定義	使用	構成ファイル
SubnetName	導入環境で使用する既存のサブネットの名前。新しいものは割り当てません。プロビジョニング解除の際に割り当て解除されることはありません。マルチリージョン導入環境の場合 (″Azure での複数のリージョンにわたる導入″ を参照)、値は指定されたリージョンごとに 1 つのコンマ区切りリストである必要があります。指定しない場合、PrivateSubnet (″セキュリティ関連のパラメータ″ を参照) が true であれば、導入環境に新しいサブネットが割り当てられ、プロビジョニング解除の際に割り当て解除されます。 注釈 プライベート・ネットワークでプロビジョニングを行う場合、定義ファイル内の各エントリについて一意の SubnetName パラメータと net_subnet_cidr パラメータを指定する必要があります (ただし、ResourceGroupName と VirtualNetworkName は既定値ファイル内で指定します)。要塞ホストを導入する場合 (″要塞ホストを使用したプライベート・ネットワークへの導入″ を参照)、これには要塞ホストの定義も含まれます。	オプション	定義ファイル
AccountTier	ストレージ・アカウントのパフォーマンス・レベル (Azure ドキュメントの ″ストレージ アカウントの概要″ を参照)。HDD (Standard) または SSD (Premium) です。	オプション	
AccountReplicationType	ストレージ・アカウントのレプリケーション・タイプ。ローカル冗長ストレージ (LRS)、geo 冗長ストレージ (GRS)、ゾーン冗長ストレージ (ZRS)、または読み取りアクセス geo 冗長ストレージ (RAGRS) です。	オプション	

テーブル 4-6: Tencent パラメータ

パラメータ	定義	使用	構成ファイル
SecretID SecretKey	Tencent Cloud アカウントを識別し、アカウントへのアクセスを提供する一意の英数字文字列。ダウンロードするには、Tencent Cloud ドキュメントの ″Signature″ を開き、“Applying for security credential” に記載されている手順に従います。	必須	既定値ファイル
ImageId	プロビジョニングされるノードのプラットフォームおよび OS のテンプレートとして使用するマシン・イメージ。Tencent ドキュメントの ″イメージの概要″ を参照してください。例 : img-pi0ii46r。	必須 (次の項目を参照)	
OSName	ImageId (前の項目を参照) が指定されていない場合、ICM は、このフィールドと一致するイメージを探します。このフィールドでは正規表現がサポートされています。既定値 : ubuntu。	必須 (前の項目を参照)	
InstanceFamily	インスタンス・タイプを選択する際に使用するインスタンス・ファミリ。InstanceType (次の項目を参照) が指定されていない場合、ICM は、InstanceFamily、CPUCoreCount、および MemorySize (後述の項目を参照) と一致するインスタンス・タイプを探します。既定値 : S3。	必須 (次の項目を参照)	

パラメータ	定義	使用	構成ファイル
InstanceType	AWS および Tencent 上にプロビジョニングされるノードの計算リソースのテンプレートとして使用するインスタンス・タイプ。Tencent ドキュメントの "インスタンス仕様" を参照してください。例 : S2.MEDIUM4。	必須 (前の項目を参照)	
ElasticIP	AWS および Tencent で Elastic IP 機能を有効にして、ホスト・ノードの再起動後も IP アドレスとドメイン名を保持します ("ホスト・ノードの再起動とリカバリ" を参照)。既定値 : false。	オプション	既定値ファイル
VPCId	AWS および Tencent への導入に使用する既存の仮想プライベート・クラウド (VPC)。新しいものは割り当てません。指定した VPC が、プロビジョニング解除の際に割り当て解除されることはありません。指定しない場合、PrivateSubnet ("セキュリティ関連のパラメータ" を参照) が true であれば、導入環境に新しい VPC が割り当てられ、プロビジョニング解除の際に割り当て解除されます。詳細は、"既存のプライベート・ネットワーク内での導入" を参照してください。 注釈 既定のアドレス空間 10.0.%d.0/24 に VPC を作成しない場合は、内部パラメータ net_subnet_cidr を指定する必要があります。例えば、範囲が 172.17.0.0/24 の VPC の場合、net_subnet_cidr を 172.17.%d.0/24 として指定する必要があります。	オプション	既定値ファイル
SubnetIds	AWS または Tencent の既存のプライベート・サブネットで導入を行う場合、サブネット ID のコンマ区切りリスト。Zone パラメータ ("一般パラメータ" を参照) によって指定された要素ごとに 1 つずつサブネット ID を追加します。	オプション	既定値ファイル
CPUCoreCount	インスタンス・タイプを選択する際に照合する CPU コア。InstanceType (前述の項目を参照) が指定されていない場合、ICM は、InstanceFamily、CPUCoreCount、および MemorySize (前述の項目を参照) と一致するインスタンス・タイプを探します。既定値 : 2。	オプション	
MemorySize	インスタンス・タイプを選択する際に照合するメモリ・サイズ。InstanceType (前述の項目を参照) が指定されていない場合、ICM は、InstanceFamily、CPUCoreCount、および MemorySize (前述の項目を参照) と一致するインスタンス・タイプを探します。既定値 : 4 GB。	オプション	
OSVolumeType	導入環境内のノードの OS ボリュームのディスク・タイプを指定します。Tencent ドキュメントの "Data Types : DataDisk" を参照してください。AWS でも同じパラメータ名が使用されます。既定値 : CLOUD_BASIC。	オプション	
DockerVolumeType	導入環境内のノードの Docker シン・プールに使用するブロック・ストレージ・デバイスのディスク・タイプを指定します。Tencent ドキュメントの "Data Types : DataDisk" を参照してください。AWS でも同じパラメータ名が使用されます。既定値 : CLOUD_BASIC。	オプション	

パラメータ	定義	使用	構成ファイル
DataVolumeType WIJVolumeType Journal1VolumeType Journal2VolumeType	**iris** コンテナ用の対応する永続ストレージ・ボリューム（"ICM によってマウントされるストレージ・ボリューム" を参照）のディスク・タイプを指定します。例えば、DataVolumeType ではデータ・ボリュームのディスク・タイプを指定します。AWS でも同じパラメータ名が使用されます。Tencent ドキュメントの "Data Types : DataDisk" を参照してください。既定値 : CLOUD BASIC。	オプション	

テーブル 4-7: vSphere パラメータ

パラメータ	定義	使用	構成ファイル
Server	vCenter サーバの名前。例 : tbdvcenter.internal.acme.com.	必須	既定値ファイル
Datacenter	データ・センタの名前。	必須	既定値ファイル
DatastoreCluster	仮想マシンのファイルを保存するデータストアのコレクション。VMware ドキュメントの "データストア クラスタの作成" を参照してください。例 : DatastoreCluster1。	必須	既定値ファイル
DataStore	設定する場合、仮想マシンのファイルを保存する、データストア・クラスタ内のデータストアを 1 つ指定します。例 : Datastore1。	オプション	既定値ファイル
ComputeCluster	計算リソース、DRS、および HA の管理に使用するホストのクラスタ。例 : ComputeCluster1。	必須	既定値ファイル
VSphereUser VSpherePassword	vSphere を操作するための資格情報。VMware ドキュメントの "About vSphere Authentication" を参照してください。	必須	既定値ファイル
DNSServers	仮想ネットワークの DNS サーバのリスト。例 : 172.16.96.1,172.17.15.53。	必須	既定値ファイル
DNSSuffixes	仮想ネットワーク・アダプタの名前解決接尾語のリスト。例 : internal.acme.com	必須	既定値ファイル
Domain	プロビジョニングされるノードの FQDN。例 : internal.acme.com	必須	既定値ファイル
NetworkInterface	ネットワーク・インタフェースに割り当てるラベル。例 : VM Network。	オプション	既定値ファイル
ResourcePool	vSphere リソース・プールの名前。VMware ドキュメントの "リソース プールの管理" を参照してください。例 : ResourcePool1。	オプション	既定値ファイル
Template	プロビジョニングされるノードのプラットフォームおよび OS のテンプレートとして使用する仮想マシンのマスタ・コピー（マシン・イメージ）。例 : ubuntu1804lts。	必須	
VCPU	プロビジョニングされるノード内の CPU 数。例 : 2。	オプション	
Memory	プロビジョニングされるノード内のメモリの量（MB 単位）。例 : 4096。	オプション	

パラメータ	定義	使用	構成ファイル
GuestID	このオペレーティング・システム・タイプ用のゲスト ID。VMware サポートの Web サイトにある "Enum – VirtualMachineGuestOsIdentifier" を参照してください。既定値 : other3xLinux64Guest。	オプション	
WaitForGuestNetTimeout	仮想マシンで使用可能な IP アドレスを待機する時間 (分単位)。既定値 : 5。	オプション	
ShutdownWaitTimeout	仮想マシンに必要な更新を行ったときの、ゲストの適切なシャットダウンまでの待機時間 (分単位)。既定値 : 3。	オプション	
MigrateWaitTimeout	仮想マシンの移行が完了するまでの待機時間 (分単位)。既定値 : 10。	オプション	
CloneTimeout	仮想マシンのクローン化が完了するまでの待機時間 (分単位)。既定値 : 30。	オプション	
CustomizeTimeout	Terraform がカスタマイゼーションの完了を待機する時間 (分単位)。既定値 : 10。	オプション	
DiskPolicy	導入のディスクのプロビジョニング・ポリシー (VMware ドキュメントの "About Virtual Disk Provisioning Policies" を参照)。値は以下のとおりです。 · thin – シン・プロビジョニング · lazy – シック・プロビジョニング (Lazy Zeroed) · eagerZeroedThick – シック・プロビジョニング (Eager Zeroed) 既定値 : lazy。	オプション	
SDRSEnabled	指定すると、仮想マシンに対してストレージ DRS (VMware ドキュメントの "ストレージ DRS の有効化と無効化" を参照) が有効になるかどうかが決まります。指定しない場合は、現在のデータストア・クラスタの設定が使用されます。既定値 : 現在のデータストア・クラスタの設定。	オプション	
SDRSAutomationLevel	これを指定すると、仮想マシンのストレージ DRS 自動化レベルが判別されます。指定しない場合は、現在のデータストア・クラスタの設定が使用されます。値は automated または manual です。既定値 : 現在のデータストア・クラスタの設定。	オプション	

パラメータ	定義	使用	構成ファイル
SDRSIntraVMAffinity	指定すると、仮想マシンの VM 内アフィニティの設定が決まります (VMware ドキュメントの "Override VMDK Affinity Rules" を参照)。指定しない場合は、現在のデータストア・クラスタの設定が使用されます。値は以下のとおりです。 · true — この仮想マシンのすべてのディスクが同一のデータストア上で保持されます。 · false — クラスタの要件を満たす場合、ストレージ DRS は複数のデータストア上に個々のディスクを配置できます。 既定値 : 現在のデータストア・クラスタの設定。	オプション	
SCSIControllerCount	指定されたホスト・ノードの SCSI コントローラの数。1 ～ 4 の数値にする必要があります。OS ボリュームは常に最初の SCSI コントローラに配置されます。vSphere では、Template フィールドによって指定されたテンプレートに定義されている数より多くの SCSI コントローラを作成できない場合があります。 既定値 : 1。	オプション	
DockerVolumeSCSICon-troller	Docker ボリュームの配置先の SCSI コントローラ。1 ～ 4 の数値にする必要があり、SCSIControllerCount より大きい数値は指定できません。 既定値 : 1。	オプション	
DataVolumeSCSIController WIJVolumeSCSIController Journal1VolumeSCSICon-troller Journal2VolumeSCSICon-troller	**iris** コンテナ内の対応するボリュームを配置する SCSI コントローラ。例えば、DataVolumeSCSIController ではデータ・ボリュームのコントローラを指定します。1 ～ 4 の数値にする必要があり、SCSIControllerCount より大きい数値は指定できません。 既定値 : 1。	オプション	

注釈 Template プロパティによって指定される VMware vSphere テンプレートの要件は、付録 “既存のクラスタへの導入” の "ホスト・ノードの要件" で説明しているものとよく似ています (例えば、パスワードを使用しない sudo アクセス)。

VMware vSphere を利用している多数のユーザのニーズに対処するために、このリリースの ICM では、vSphere がサポートされています。特有の vSphere 構成と基盤のハードウェア・プラットフォームに応じて、ICM を使用して仮想マシンをプロビジョニングするには、このガイドに記載されていない追加の拡張や調整が必要になることがあります。大規模で複雑な導入の場合は特に、その可能性があります。また、ICM を使用した仮想マシンのプロビジョニングがプロダクションでの使用に適さないこともあります。完全なサポートは、今後のリリースで予定されています。

4.2.6 デバイス名パラメータ

以下に示すパラメータでは、/dev に配置されるデバイス・ファイルを指定します。これらのファイルは、ICM によって作成され、InterSystems IRIS によって使用される永続ボリュームを表します。これらの永続ボリュームと、これらのパラメータのプロバイダおよび OS 固有の既定値のテーブルについては、"ICM によってマウントされるストレージ・ボリューム" を参照してください。PreExisting の導入については、付録 "既存のクラスタへの導入" の "ストレージ・ボリューム" を参照してください。

パラメータ	永続ボリュームの用途
DataDeviceName	データベース
WIJDeviceName	WIJ ディレクトリ
Journal1DeviceName	プライマリ・ジャーナル・ディレクトリ
Journal2DeviceName	代替ジャーナル・ディレクトリ

4.2.7 ユーザ・パラメータのアルファベット順のリスト

以下のテーブルは、このセクションの前述のテーブルで説明したすべてのパラメータをアルファベット順にリストしたものです。それぞれの定義を含むテーブルへのリンクも用意されています。

パラメータ	定義のテーブル
AccountReplicationType	プロバイダ固有 – Azure
AccountTier	プロバイダ固有 – Azure
AlternativeServers	一般
AMI	プロバイダ固有 – AWS
ApplicationPath	一般
ClientId	プロバイダ固有 – Azure、セキュリティ
ClientSecret	プロバイダ固有 – Azure、セキュリティ
CloneTimeout	プロバイダ固有 – vSphere
ComputeCluster	プロバイダ固有 – vSphere
Count	一般
CPUCoreCount	プロバイダ固有 – Tencent
Credentials	プロバイダ固有 – AWS、プロバイダ固有 – GCP、セキュリティ
CustomizeTimeout	プロバイダ固有 – vSphere
Datacenter	プロバイダ固有 – vSphere

パラメータ	定義のテーブル
DataDeviceName	デバイス名
DataMountPoint	一般
Datastore	プロバイダ固有 – vSphere
DatastoreCluster	プロバイダ固有 – vSphere
DataVolumeIOPS	プロバイダ固有 – AWS
DataVolumeSCSIController	プロバイダ固有 – vSphere
DataVolumeSize	一般
DataVolumeType	プロバイダ固有 – AWS、プロバイダ固有 – GCP、プロバイダ固有 – Tencent
DiskPolicy	プロバイダ固有 – vSphere
DNSName	PreExisting
DNSServers	プロバイダ固有 – vSphere
DNSSuffixes	プロバイダ固有 – vSphere
DockerImage	一般
DockerInit	一般
DockerPassword	一般
DockerRegistry	一般
DockerStorageDriver	一般
DockerURL	一般
DockerUsername	一般
DockerVersion	一般
DockerVolumeIOPS	プロバイダ固有 – AWS
DockerVolumeSCSICon-troller	プロバイダ固有 – vSphere
DockerVolumeSize	一般
DockerVolumeType	プロバイダ固有 – AWS、プロバイダ固有 – GCP、プロバイダ固有 – Tencent
Domain	プロバイダ固有 – vSphere

パラメータ	定義のテーブル
ElasticIP	プロバイダ固有 – AWS、プロバイダ固有 – Tencent
FileSystem	一般
GuestID	プロバイダ固有 – vSphere
Image	プロバイダ固有 – GCP
ImageId	プロバイダ固有 – Tencent
InstanceFamily	プロバイダ固有 – Tencent
InstanceType	プロバイダ固有 – AWS、プロバイダ固有 – Tencent
InternetGatewayId	プロバイダ固有 – AWS
IPAdress	PreExisting
ISCPassword	一般
Journal1DeviceName	デバイス名
Journal1MountPoint	一般、CPF
Journal1VolumeIOPS	プロバイダ固有 – AWS
Journal1VolumeSCSICon-troller	プロバイダ固有 – vSphere
Journal1VolumeSize	一般
Journal1VolumeType	プロバイダ固有 – AWS、プロバイダ固有 – GCP、プロバイダ固有 – Tencent
Journal2DeviceName	デバイス名
Journal2MountPoint	一般、CPF
Journal2VolumeIOPS	プロバイダ固有 – AWS
Journal2VolumeSCSICon-troller	プロバイダ固有 – vSphere
Journal2VolumeSize	一般
Journal2VolumeType	プロバイダ固有 – AWS、プロバイダ固有 – GCP、プロバイダ固有 – Tencent
Label	一般
LicenseDir	一般
LicenseKey	一般

パラメータ	定義のテーブル
LicenseServerPort	ポート、CPF
LoadBalancer	一般
LoadBalancerInternal	プロバイダ固有 – AWS
Location	プロバイダ固有 – Azure
LocationMap	プロバイダ固有 – Azure
MachineType	プロバイダ固有 – GCP
Memory	プロバイダ固有 – vSphere
MemorySize	プロバイダ固有 – Tencent
MigrateWaitTimeout	プロバイダ固有 – vSphere
Mirror	一般
MirrorMap	一般
Namespace	一般
NetworkInterface	プロバイダ固有 – vSphere
OSName	プロバイダ固有 – Tencent
OSVolumeIOPS	プロバイダ固有 – AWS
OSVolumeSize	一般
OSVolumeType	プロバイダ固有 – AWS、プロバイダ固有 – GCP、プロバイダ固有 – Tencent
Overlay	一般
PlacementGroups	プロバイダ固有 – AWS
PlacementStrategy	プロバイダ固有 – AWS
PlacementMap	プロバイダ固有 – AWS
PlacementPartitionCount	プロバイダ固有 – AWS
PlacementSpreadLevel	プロバイダ固有 – AWS
Project	プロバイダ固有 – GCP
Provider	一般
ProxyImage	一般

パラメータ	定義のテーブル
Region	一般
RegionMap	プロバイダ固有 – GCP
ResourceGroupName	プロバイダ固有 – Azure
ResourcePool	プロバイダ固有 – vSphere
Role	一般
RouteTableId	プロバイダ固有 – AWS
SCSIControllerCount	プロバイダ固有 – vSphere
SDRSAutomationLevel	プロバイダ固有 – vSphere
SDRSEnabled	プロバイダ固有 – vSphere
SDRSIntraVMAffinity	プロバイダ固有 – vSphere
SecretID	プロバイダ固有 – Tencent、セキュリティ
SecretKey	プロバイダ固有 – Tencent、セキュリティ
Server	プロバイダ固有 – vSphere
ShutdownWaitTimeout	プロバイダ固有 – vSphere
Size	プロバイダ固有 – Azure
SSHOnly	セキュリティ
SSHPassword	セキュリティ
SSHPrivateKey	セキュリティ
SSHPublicKey	セキュリティ
SSHUser	セキュリティ
SSLConfig	セキュリティ
StartCount	一般
SubnetName	プロバイダ固有 – Azure
SubnetIds	プロバイダ固有 – AWS、プロバイダ固有 – Tencent
SubscriptionId	セキュリティ
SuperServerPort	ポート、CPF

パラメータ	定義のテーブル
SystemMode	一般
Tag	一般
Template	プロバイダ固有 – vSphere
TenantId	セキュリティ
TLSKeyDir	セキュリティ
UseMSI	プロバイダ固有 – Azure、セキュリティ
UserCPF	一般
VCPU	プロバイダ固有 – vSphere
VirtualNetworkName	プロバイダ固有 – Azure
VPCId	プロバイダ固有 – AWS、プロバイダ固有 – Tencent
VspherePassword	プロバイダ固有 – vSphere、セキュリティ
VsphereUser	プロバイダ固有 – vSphere、セキュリティ
WaitForGuestNetTimeout	プロバイダ固有 – vSphere
WeavePassword	セキュリティ
WebGatewayPort	ポート
WebServerPort	ポート、CPF
WIJDeviceName	デバイス名
WIJMountPoint	一般、CPF
WIJVolumeIOPS	プロバイダ固有 – AWS
WIJVolumeSCSIController	プロバイダ固有 – vSphere
WIJVolumeSize	一般
WIJVolumeType	プロバイダ固有 – AWS、プロバイダ固有 – GCP、プロバイダ固有 – Tencent
Zone	一般
ZoneMap	一般

4.3 ICM ノード・タイプ

このセクションでは、ICM によってプロビジョニングおよび導入できるノードのタイプと、導入される InterSystems IRIS 構成で可能なロールについて説明します。プロビジョニングされるノードのタイプは、[**ロール**] フィールドによって決まります。

以下のテーブルにノード・タイプの要約を示し、その後で詳しく説明します。

テーブル 4-8: ICM ノード・タイプ

ノード・タイプ	構成ロール	導入するインターシステムズのイメージ
DATA	シャード・クラスタ・データ・ノード	iris (InterSystems IRIS インスタンス)
COMPUTE	シャード・クラスタ計算ノード	iris (InterSystems IRIS インスタンス)
DM	分散キャッシュ・クラスタ・データ・サーバ スタンドアロンの InterSystems IRIS インスタンス [ネームスペース・レベルのアーキテクチャ : シャード・マスタ・データ・サーバ]	iris (InterSystems IRIS インスタンス)
DS	[ネームスペース・レベルのアーキテクチャ : シャード・データ・サーバ]	iris (InterSystems IRIS インスタンス)
QS	[ネームスペース・レベルのアーキテクチャ : シャード・クエリ・サーバ]	iris (InterSystems IRIS インスタンス)
AM	分散キャッシュ・クラスタ・アプリケーション・サーバ	iris (InterSystems IRIS インスタンス)
AR	ミラー・アービター	arbiter (InterSystems IRIS ミラー・アービター)
WS	Web サーバ	webgateway (インターシステムズの Web ゲートウェイ)
SAM	System Alerting and Monitoring (SAM) ノード	sam (InterSystems System Alerting and Monitoring アプリケーション)
LB	ロード・バランサ	—
VM	仮想マシン	—
CN	カスタム・コンテナ・ノードおよびサードパーティ・コンテナ・ノード	—
BH	要塞ホスト	—

重要

上のテーブルに示されているインターシステムズのイメージは、対応するノード・タイプで必須であり、対応しないノードに導入することはできません。**DockerImage** フィールドまたは icm run コマンドの -image オプションでノードについて誤ったインターシステムズのイメージが指定された場合 (例えば、**iris** イメージが AR (アービター) ノードについて指定された場合や、インターシステムズのいずれかのイメージが CN ノードについて指定された場合)、導入は失敗し、ICM から該当するメッセージが返されます。インターシステムズのイメージの導入の詳細は、“ICM の使用” の章にある ″icm run コマンド″ を参照してください。

注釈　上のテーブルには、このドキュメントの以前のバージョンに記載されていたネームスペース・レベルのシャーディング・アーキテクチャのシャード・クラスタのロールが含まれています。これらのロール (DM、DS、QS) は引き続き ICM で使用できますが、同じ導入環境内で DATA ノードや COMPUTE ノードと組み合わせることはできません。

4.3.1 ロール DATA：シャード・クラスタ・データ・ノード

″スケーラビリティ・ガイド″ の “シャーディングによるデータ量に応じた水平方向の拡張” の章の ″InterSystems IRIS のシャーディングの概要″ などのセクションに記載されているように、一般的なシャード・クラスタはデータ・ノードのみで構成され、それらのノード間でシャード・データが分割されるため、definitions.json ファイルに必要なのは DATA ノードの定義のみです。DATA ノードを定義する場合、導入環境はシャード・クラスタである必要があり、DATA ノードと共に定義できる他のノード・タイプは COMPUTE のみです。

ノード定義の MirrorMap の設定と一致する数でプロビジョニングする場合、DATA ノードをミラーリングすることができます (″ミラーリングの規則″ を参照)。クラスタ内の DATA ノードは、すべてをミラーリングするか、すべてをミラーリングしないかのどちらかにする必要があります。

シャード・クラスタ内のデータ・ノード間の唯一の相違は、最初に構成されたノード (ノード 1 と呼ばれる) には、そのノードへのシャード・データの割り当て分に加えて、クラスタのシャード化されていないデータ、メタデータ、およびコードがすべて格納される点です。ただし、ストレージ要件の違いは一般的にごくわずかです。すべてのデータ、メタデータ、およびコードをクラスタ内のいずれのノードでも認識できるので、すべてのノード間でアプリケーション接続を負荷分散して、クエリの並列処理や分割されたキャッシュを最大限に活用することができます。ロード・バランサを DATA ノードに割り当てることができます。″ロール LB：ロード・バランサ″ を参照してください。

4.3.2 ロール COMPUTE：シャード・クラスタ計算ノード

常に大量のデータが取り込まれるような状況でも、クエリの遅延がきわめて小さいことが求められる高度な使用事例では、計算ノードをシャード・クラスタに追加して、クエリを処理するための透過的なキャッシュ層を提供することで、クエリとデータ取り込みの作業負荷を分離し、両方のパフォーマンスを向上させることができます (詳細は、″スケーラビリティ・ガイド″ の ″作業負荷の分離とクエリ・スループットの向上のための計算ノードの導入″ を参照してください)。

計算ノードを追加することによってパフォーマンスが大幅に向上するのは、データ・ノードごとに計算ノードが少なくとも 1 つある場合のみです。したがって、少なくとも DATA ノードと同数の COMPUTE ノードを定義する必要があります。定義ファイル内の DATA ノードの数が COMPUTE ノードの数よりも多い場合、ICM は警告を発行します。1 つのデータ・ノードに対して複数の計算ノードを構成すると、クラスタのクエリ・スループットをさらに向上させることができます。その際、ベスト・プラクティスとして、ICM が定義された COMPUTE ノードを DATA ノード全体にできるだけ均等に分散できるように、各データ・ノードの計算ノードの数が同じになるように構成することをお勧めします。

COMPUTE ノードではクエリの実行のみがサポートされ、データは格納されないので、メモリと CPU を重視し、ストレージを最小限に抑えるなど、ニーズに合わせてインスタンス・タイプやその他の設定を調整できます。COMPUTE ノードにはデータが格納されないので、ミラーリングすることはできません。

ロード・バランサを COMPUTE ノードに割り当てることができます。″ロール LB：ロード・バランサ″ を参照してください。

4.3.3 ロール DM：分散キャッシュ・クラスタ・データ・サーバ、スタンドアロン・インスタンス、シャード・マスタ・データ・サーバ

ロール AM の複数のノードと DM ノード (ミラーリングなし、またはあり) が指定された場合、これらは InterSystems IRIS 分散キャッシュ・クラスタとして導入され、前者はアプリケーション・サーバ、後者はデータ・サーバとして動作します。

単独で導入されるロール DM のノード (ミラーリングなし、またはあり) は、スタンドアロン InterSystems IRIS インスタンスになります。

DM ノード (ミラーリングあり、またはミラーリングなし)、DS ノード (ミラーリングあり、またはミラーリングなし)、および (オプションで) QS ノードが指定されている場合、これらはネームスペース・レベルのシャード・クラスタとして導入されます。

4.3.4 ロール DS : シャード・データ・サーバ

ネームスペース・レベルのアーキテクチャでは、シャード・クラスタにロードされた各シャード・テーブルの行分割がデータ・シャードに 1 つ格納されます。データ・シャードをホストするノードは、シャード・データ・サーバと呼ばれます。1 つのクラスタに複数 (最多で 200 台超) のシャード・データ・サーバを含めることができます。偶数台のシャード・データ・サーバを導入しミラーリングを指定することで、シャード・データ・サーバをミラーリングできます。

4.3.5 ロール QS : シャード・クエリ・サーバ

ネームスペース・レベルのアーキテクチャでは、シャード・クエリ・サーバが、割り当てられたデータ・シャードへのクエリ・アクセスを提供することにより、クエリとデータ取り込みの作業負荷間の干渉を最小限に抑え、マルチユーザによる大量のクエリ作業負荷を処理するシャード・クラスタの処理能力を高めます。シャード・クエリ・サーバが導入されている場合、導入されたシャード・データ・サーバにラウンドロビン方式で割り当てられます。ミラーリングされたシャード・データ・サーバのフェイルオーバー時に、シャード・クエリ・サーバはアプリケーションの接続を自動的にリダイレクトします。

QS ノードは定義されているが DS ノードが定義されていない場合、ICM は以下のようなエラーで応答します。

```
Shard Query Server (role 'QS') requires at least one Shard Data Server (role 'DS')
```

4.3.6 ロール AM : 分散キャッシュ・クラスタ・アプリケーション・サーバ

ロール AM の複数のノードと DM ノードが指定された場合、これらは InterSystems IRIS 分散キャッシュ・クラスタとして導入され、前者はアプリケーション・サーバ、後者はデータ・サーバとして動作します。データ・サーバがミラーリングされている場合は、フェイルオーバー後のアプリケーション接続のリダイレクトが自動的に行われます。

ロード・バランサを AM ノードに割り当てることができます。"ロール LB : ロード・バランサ" を参照してください。

4.3.7 ロール AR : ミラー・アービター

DATA ノード (シャード・クラスタ DATA ノード)、DM ノード (分散キャッシュ・クラスタ・データ・サーバ、スタンドアロン InterSystems IRIS インスタンス、またはネームスペース・レベルのシャード・マスタ・データ・サーバ)、または DS ノード (ネームスペース・レベルのシャード・データ・サーバ) がミラーリングされている場合は、自動フェイルオーバーを円滑に行うためにアービター・ノードを導入することを強くお勧めします。クラスタ内のすべてのミラーに対してアービター・ノードは 1 つあれば十分で、複数のアービターはサポートされておらず、ICM によって無視されます (ミラーリングされていないクラスタ内のアービター・ノードも同様に無視されます)。

AR ノードは InterSystems IRIS インスタンスを含んでおらず、別のイメージを使用して ISCAgent コンテナを実行します。この ・イメージは、AR ノードの定義ファイル・エントリ内の **DockerImage** フィールドを使用して指定する必要があります。詳細は、"icm run コマンド" を参照してください。

アービターの詳細は、"高可用性ガイド" の “ミラーリング” の章を参照してください。

4.3.8 ロール WS : Web サーバ

1 つの導入環境に Web サーバをいくつでも含めることができます。各 Web サーバ・ノードには、Apache Web サーバと共にインターシステムズの Web ゲートウェイのインストールが含まれます。ICM は、インターシステムズの Web ゲートウェイのリモート・サーバ・リストを以下のように生成します。

- DATA ノードと COMPUTE ノードを導入する場合 (ノード・レベルのシャード・クラスタ)、すべての DATA ノードと COMPUTE ノード。COMPUTE ノードを導入しない場合は、すべての DATA ノード。

- AM ノードを導入する場合 (分散キャッシュ・クラスタ)、すべての AM ノード。
- それ以外の場合は、DM ノード (スタンドアロン・インスタンスまたはネームスペース・レベルのシャード・クラスタ)。

注釈　Web サーバ層でネームスペース・レベルのシャード・クラスタを導入する場合、カスタムまたはサードパーティのロード・バランサを手動で導入して、クラスタ内の DS ノードおよび (存在する場合) QS ノードの間で接続を分散し (″スケーラビリティ・ガイド″ の ″ネームスペース・レベル・アーキテクチャの導入″ で推奨されているとおり)、Web ゲートウェイ構成を手動で編集して、リモート・サーバのリストにロード・バランサのアドレスを入力できます (詳細は、″Web ゲートウェイ構成ガイド″ の ″ミラー構成、フェイルオーバー、および負荷分散″ を参照)。

ミラーリングされた DATA ノードと DM ノードについては、ミラー認識接続が作成され、フェイルオーバー後のアプリケーション接続のリダイレクトが自動的に行われます。Web サーバとリモート・サーバの間の通信は TLS モードで動作するように構成されます。

ロード・バランサを WS ノードに割り当てることができます。″ロール LB : ロード・バランサ″ を参照してください。

WS ノードは InterSystems IRIS インスタンスを含んでおらず、別のイメージを使用して Web ゲートウェイ・コンテナを実行します。″icm run コマンド″ で説明しているように、**webgateway** イメージは、definitions.json ファイルの WS ノード定義に DockerImage フィールドを含めることで指定できます。以下に例を示します。

```
{
    "Role": "WS",
    "Count": "3",
    "DockerImage": "intersystems/webgateway:latest-em",
    "ApplicationPath": "/acme",
    "AlternativeServers": "LoadBalancing"
}
```

ApplicationPath フィールドを指定する場合、その値は、Web ゲートウェイのインスタンスごとのアプリケーション・パスの作成に使用されます。このアプリケーション・パスの既定のサーバは、ラウンドロビン方式で Web ゲートウェイ・インスタンス間で割り当てられ、それ以外のリモート・サーバは代替サーバ・プールを構成します。例えば、前の例の WS ノード定義が 3 つの AM ノードの分散キャッシュ・クラスタの一部である場合、割り当ては以下のようになります。

インスタンス	既定のサーバ	代替サーバ
Acme-WS-TEST-0001	Acme-AM-TEST-0001	Acme-AM-TEST-0002、Acme-AM-TEST-0003
Acme-WS-TEST-0002	Acme-AM-TEST-0002	Acme-AM-TEST-0001、Acme-AM-TEST-0003
Acme-WS-TEST-0003	Acme-AM-TEST-0003	Acme-AM-TEST-0001、Acme-AM-TEST-0002

AlternativeServers フィールドには、Web ゲートウェイがターゲット・サーバ・プールに要求をどのように分散するかを指定します。有効な値は、LoadBalancing (既定) と FailOver です。ApplicationPath フィールドを指定しない場合、このフィールドは無効です。

インターシステムズの Web ゲートウェイの使用法の詳細は、″Web ゲートウェイ構成ガイド″ を参照してください。

4.3.9 ロール SAM : System Alerting and Monitoring ノード

SAM ノードを定義すると、System Alerting and Monitoring (SAM) クラスタ・モニタリング・ソリューションが導入環境に追加されます。SAM の追加の詳細は、″ICM でのモニタリング″、SAM の詳細は、″System Alerting and Monitoring Guide″ を参照してください。

4.3.10 ロール LB : ロード・バランサ

プロビジョニング・プラットフォームが AWS、GCP、Azure、または Tencent であり、定義ファイル内のタイプ DATA、COMPUTE、DM、AM、または WS のノードの定義で LoadBalancer が true に設定されている場合、ICM は、事前定義されたロード・バランサ・ノードを自動的にプロビジョニングします。VM ノードまたは CN ノードの汎用ロード・バランサについては、追加のパラメータを指定する必要があります。

4.3.10.1 事前定義されたロード・バランサ

ロール LB のノードの場合、ICM は転送するポートとプロトコル、および対応する正常性チェックを構成します。導入されたロード・バランサに対するクエリは、シャード・クラスタ内のデータ・ノードまたは分散キャッシュ・クラスタ・アプリケーション・サーバに対して行う場合と同じように実行できます。

DATA、COMPUTE、DM、AM、または WS のノードの定義にロード・バランサを追加するには、LoadBalancer フィールドを追加します。以下に例を示します。

```
{
    "Role": "AM",
    "Count": "2",
    "LoadBalancer": "true"
}
```

以下の例は、この定義によって作成および導入されるノードを示しています。

```
$ icm inventory
Machine           IP Address    DNS Name                                Region   Zone
-------           ----------    --------                                ------   ----
ANDY-AM-TEST-0001 54.214.230.24 ec2-54-214-230-24.amazonaws.com         us-west1 c
ANDY-AM-TEST-0002 54.214.230.25 ec2-54-214-230-25.amazonaws.com         us-west1 c
ANDY-LB-TEST-0000 (virtual AM)  ANDY-AM-TEST-1546467861.amazonaws.com us-west1 c
```

このクラスタに対するクエリは、AM ノードに対して行う場合と同じように、ロード・バランサに対して実行できます。

ミラーリングした DATA ノードと DM ノードに事前定義したロード・バランサはミラー認識ロード・バランサとなり、必ず現在のプライマリにトラフィックを転送します。

LoadBalancer フィールドは、1 つの導入環境の複数の定義に追加できます。例えば、負荷分散されたアプリケーション接続を受け取る WS 層から負荷分散された接続を受け取る AM ノードを分散キャッシュ・クラスタに含めることができます。

現在、自動的にプロビジョニングされた単一のロード・バランサを複数のノード・タイプ（例えば、DATA ノードと COMPUTE ノードの両方）で共有することはできないため、それぞれに専用のロード・バランサが必要です。目的のロール用にユーザがカスタムまたはサードパーティのロード・バランサを手動で導入してもかまいません。別の有用なアプローチは、WS ノードのロード・バランサをプロビジョニングすることです。これにより、"ロール WS : Web サーバ" で説明しているように、アプリケーションの接続を複数のノード・タイプに分散できます。

注釈　AWS でプロビジョニングする場合、LoadBalancer を True に設定した定義で LoadBalancerInternal を True に設定することにより、"internal" タイプのロード・バランサを指定できます。

4.3.10.2 汎用ロード・バランサ

以下の追加キーを指定することにより、ロード・バランサを VM（仮想マシン）ノードおよび CN（コンテナ）に追加できます。

- ForwardProtocol
- ForwardPort
- HealthCheckProtocol
- HealthCheckPath

- HealthCheckPort

以下に例を示します。

```
{
    "Role": "VM",
    "Count": "2",
    "LoadBalancer": "true",
    "ForwardProtocol": "tcp",
    "ForwardPort": "443",
    "HealthCheckProtocol": "http",
    "HealthCheckPath": "/csp/status.cxw",
    "HealthCheckPort": "8080"
}
```

ForwardPort には、転送するポートのコンマ区切りリストを指定できます。その場合、転送するポートすべてで同じ ForwardProtocol を共有していることが条件になります。

これらのキーの詳細は、“ICM の構成パラメータ” の ″ポートおよびプロトコルのパラメータ″ のテーブルを参照してください。

4.3.10.3 ロード・バランサに関する注意事項

それぞれのクラウド・プロバイダ上のロード・バランサは、異なる動作をすることがあります。必ず、プロビジョニングするプラットフォームにおけるロード・バランサの詳細を十分に理解してください。特に次のことに留意してください。

- クラウド・プロバイダによっては、複数の IP アドレスに解決される DNS 名をロード・バランサ用に作成している場合があります。この理由から、[DNS Name] 列の値を使用する必要があります。[DNS Name] 列に数値 IP アドレスが表示されている場合は、単にそのクラウド・プロバイダがロード・バランサに一意の IP アドレスを割り当てていて、DNS 名は指定していないことを示しています。
- その DNS 名は特定のロード・バランサが適用されるリソースを示していない可能性があるので、[IP Address] 列がこの目的に使用されます。
- ターゲット・プールのすべてのメンバが正常性チェックに失敗した場合の対応方法は、クラウド・プロバイダごとに異なる場合があります。インターシステムズが確認したところでは、GCP の場合は要求がランダムなターゲット (基盤となるサービスが使用できる場合と使用できない場合がある) に転送されるのに対し、AWS ではロード・バランサが要求を拒否します。

プロバイダ Vmware vSphere および PreExisting の場合は、カスタムまたはサードパーティのロード・バランサを導入することをお勧めします。

ミラーリングされた DATA ノード用のロード・バランサを Tencent にプロビジョニングしないでください。Tencent にプロビジョニングされたロード・バランサは現在、ミラーリングされた DATA ノードのどちらがプライマリであるかを判別できないため、ロード・バランサを介した読み取り/書き込み操作の実行時にエラーが発生することがあります。

4.3.11 ロール VM : 仮想マシン・ノード

クラスタには仮想マシン・ノードをいくつでも含めることができます。仮想マシン・ノードは、InterSystems IRIS クラスタ内で事前定義されたロールを持たないホスト・ノードを割り振るための手段です。これらのノードに Docker はインストールされませんが、ユーザは任意のカスタムまたはサードパーティのソフトウェア (Docker を含む) を自由に導入できます。

仮想マシン・ノードでは以下のコマンドがサポートされます。

- icm provision
- icm unprovision
- icm inventory
- icm ssh

- icm scp

ロード・バランサを VM ノードに割り当てることができます。"ロール LB : ロード・バランサ" を参照してください。

4.3.12 ロール CN : コンテナ・ノード

クラスタにはコンテナ・ノードをいくつでも含めることができます。コンテナ・ノードは、Docker がインストールされている汎用ノードです。

CN ノードを定義して、**IAM** および **IAMImage** フィールドを含めることで、InterSystems API Manager (IAM) を任意の導入環境に追加できます。詳細は、"ICM リファレンス" の "InterSystems API Manager の導入" を参照してください。任意のカスタム・コンテナやサードパーティ・コンテナも CN ノードに導入できます。指定されている場合、**iris** (InterSystems IRIS) コンテナは導入されません。コンテナ・ノードではすべての ICM コマンドがサポートされていますが、-container オプションを使用して **iris** 以外のコンテナを指定する場合や、-role オプションまたは -machine オプションを使用して CN ノードに対するコマンドに限定する場合を除き、ほとんどの ICM コマンドがフィルタで除外されます ("ICM コマンドとオプション" を参照)。

ロード・バランサを CN ノードに割り当てることができます。"ロール LB : ロード・バランサ" を参照してください。CN ノードはコンテナレス・モードでは導入できません。

4.3.13 ロール BH : 要塞ホスト

パブリック・ネットワーク・アクセスを提供しない構成を導入することもできます。既存のプライベート・ネットワークがある場合、そのネットワーク上のノードで ICM を起動し、その中で導入を行うことができます。そのようなネットワークがない場合は、ICM によってプライベート・サブネットが構成され、そこに構成が導入されるようにすることができます。ただし、ICM はそのプライベート・サブネット内で実行されていないので、構成のプロビジョニング、導入、および管理を行うためのアクセス手段が必要です。この目的を果たすのが BH ノードです。

要塞ホストは、ICM によって構成されたプライベート・サブネットとパブリック・ネットワークの両方に属し、それらの間の通信を仲介できるホスト・ノードです。要塞ホストを使用するには、定義ファイルで 1 つの BH ノードを定義し、既定値ファイルで **PrivateSubnet** を true に設定します。詳細は、"プライベート・ネットワークへの導入" を参照してください。

4.4 ICM クラスタのトポロジとミラーリング

ICM は定義ファイル内のノード定義を検証して、特定の要件を満たしているかどうか確認します。ミラーリングされた構成には、追加の規則があります。この検証には、機能面で最適ではない構成 (例えば、単一の AM ノード、単一の WS ノード、5 つの DATA ノードに対してただ 1 つの COMPUTE ノード、またはその逆の構成など) を回避する処理は含まれないことに注意してください。

ミラーリングされていない構成とミラーリングされた構成の両方で、以下のように動作します。

- シャード・クラスタでは、COMPUTE ノードはラウンドロビン方式で DATA ノードに割り当てられます (DS ノードへの QS ノードの割り当ても同様です)。
- AM ノードと WS ノードの両方が含まれている場合、AM ノードは DM ノードにバインドされ、WS ノードは AM ノードにバインドされます。AM ノードのみまたは WS ノードのみが含まれている場合は、すべてのノードが DM ノードにバインドされます。

このセクションには、以下のサブセクションがあります。

- ミラーリングの規則
- ミラーリングされていない構成の要件

- ミラーリングされた構成の要件

4.4.1 ミラーリングの規則

シャード・クラスタ内のデータ・ノードは、すべてをミラーリングするか、すべてをミラーリングしないかのどちらかにする必要があります。この要件は、以下の ICM トポロジ検証規則に反映されています。

既定値ファイルで Mirror フィールドが false に設定されている場合 (既定値)、ミラーリングは構成されず、定義ファイルに複数の DM ノードが指定されているとプロビジョニングは失敗します。

Mirror フィールドが true に設定されている場合、可能であればミラーリングが構成され、DATA、DS、または DM ノードのミラー・ロール (プライマリ、バックアップ、または DR 非同期) がノード定義の MirrorMap フィールドの値 (″一般パラメータ″ を参照) によって、次のように決定されます。

- MirrorMap が関連するノード定義に含まれていない場合、ノードは MirrorMap の既定の値 primary,backup を使用して、ミラー・フェイルオーバー・ペアとして構成されます。
  - 偶数個の DATA ノードまたは DS ノードが定義されている場合、これらはすべてフェイルオーバー・ペアとして構成されます。例えば、6 つの DATA ノードを指定すると、フェイルオーバー・ペアを含み、DR 非同期を含まない 3 つのデータ・ノード・ミラーが導入されます。奇数個の DATA ノードまたは DS ノードが定義されている場合、プロビジョニングは失敗します。
  - 2 つの DM ノードが定義されている場合、これらはフェイルオーバー・ペアとして構成されます。それ以外の数が定義されている場合、プロビジョニングは失敗します。

- ノード定義に MirrorMap が含まれている場合、ノードはその値に基づいて次のように構成されます。
  - DATA ノードまたは DS ノードの個数は、MirrorMap の値で指定されたロールの数以下の倍数である必要があります。例えば、次に示すように、MirrorMap の値が primary,backup,async であるとします。

    ```
    "Role": "DATA",
    "Count": "",
    "MirrorMap": "primary,backup,async"
    ```

    この場合、DATA ノードまたは DS ノードは次のように構成されます。

Count の値	結果
3 または 3 の倍数	フェイルオーバー・ペアと DR 非同期を含む 1 つ以上のミラー
2	フェイルオーバー・ペアを含む 1 つのミラー
1、4 以上 (3 の倍数以外)	プロビジョニング失敗

  - DM ノードの数は、MirrorMap の値で指定されたロールの数以下である必要があります。DM ノードを 1 つだけ指定すると、プロビジョニングは失敗します。

- 複数の AR (アービター) ノードが指定されている場合、プロビジョニングは失敗します。(ベスト・プラクティスではありますが、アービターの使用は任意であるため、ミラー構成に AR ノードを含める必要はありません。)

ICM によって導入される非同期はすべて DR 非同期です。レポート非同期はサポートされていません。1 つのミラーに最大 14 個の非同期を含めることができます。ミラー・メンバと可能な構成の詳細は、″高可用性ガイド″ の ″ミラー・コンポーネント″ を参照してください。

DATA ノード、DS ノード、または DM ノードのプロビジョニングや構成が行われた順序とミラー内でのそれらのロールとの間に関係性はありません。プロビジョニングの後、icm inventory コマンドを使用して、各ペアのどのメンバがプライマ

リ・フェイルオーバー・メンバとなり、どのメンバがバックアップとなる予定かを確認できます。ミラーリングが有効な場合に、導入された構成で各ノードのミラー・メンバ・ステータスを確認するには、icm ps コマンドを使用します。

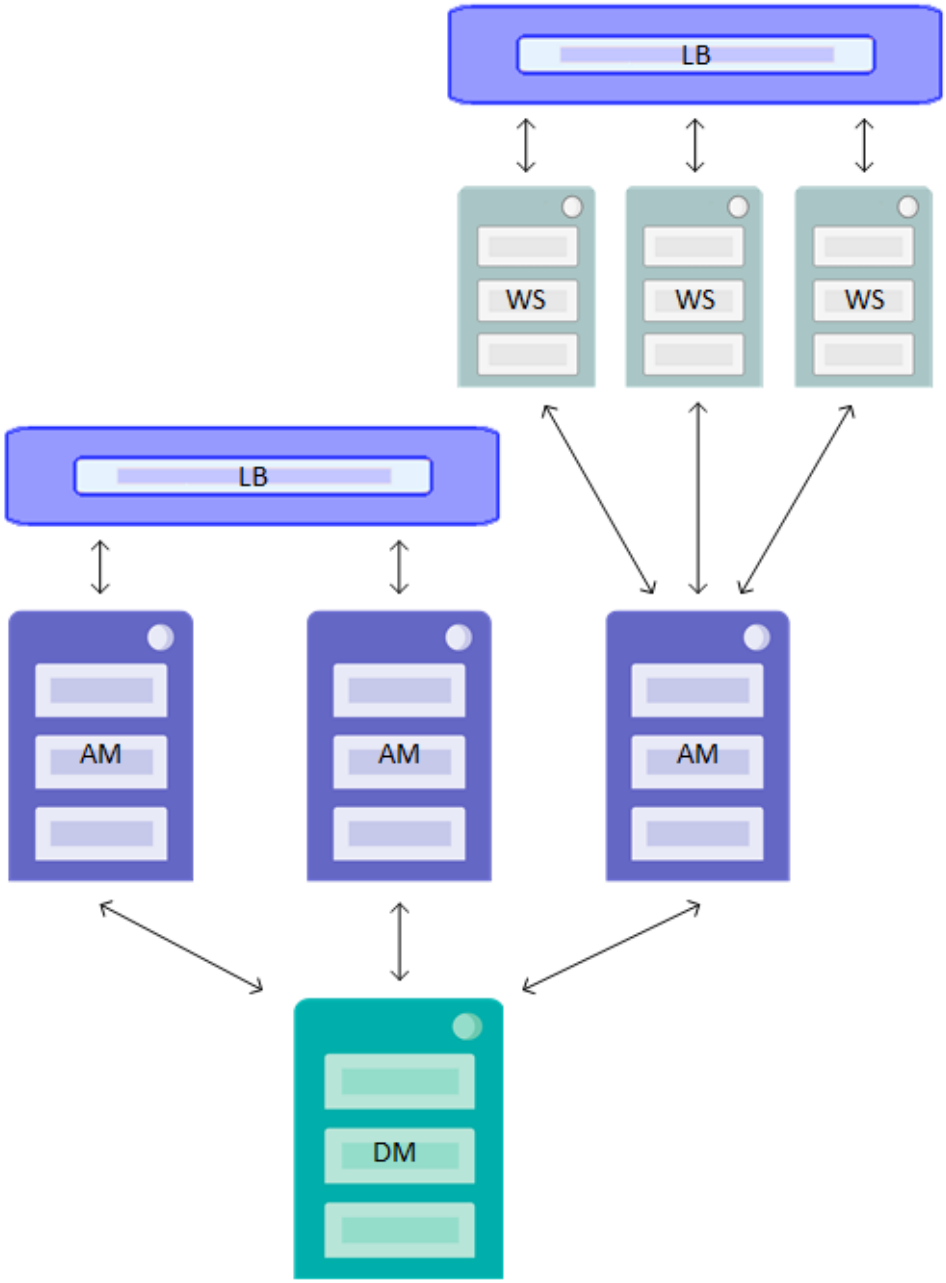
4.4.2 ミラーリングされていない構成の要件

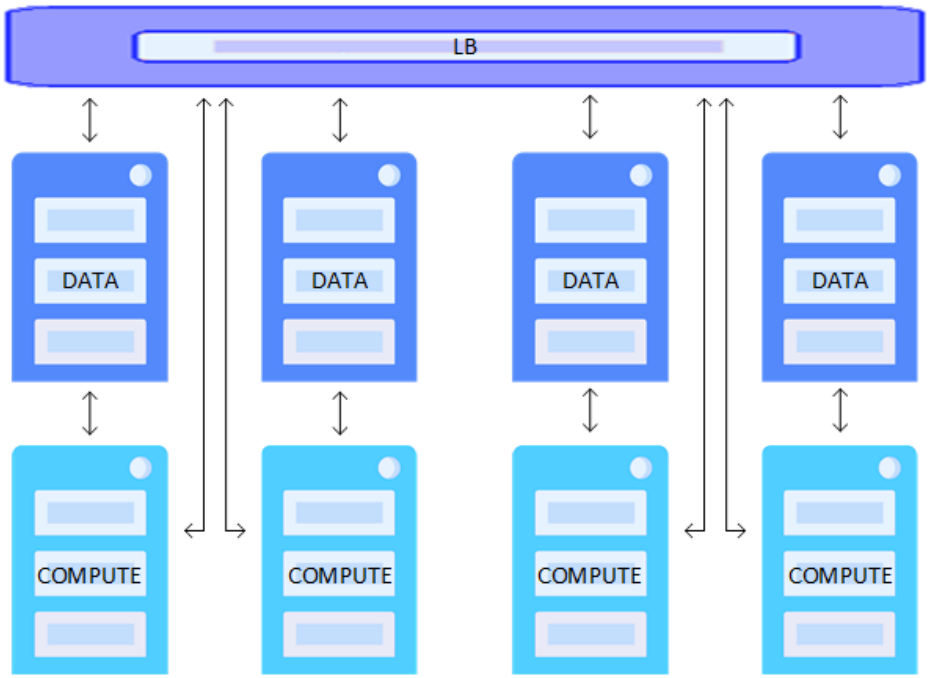
ミラーリングされていないクラスタは以下で構成されます。

- 1 つ以上の DATA (シャード・クラスタ内のデータ・ノード)。
- DATA ノードが含まれている場合、ゼロ個以上の COMPUTE (シャード・クラスタ内の計算ノード)。ベスト・プラクティスは、少なくとも DATA ノードと同数の COMPUTE ノードを含め、各 DATA ノードの COMPUTE ノードの数を同じにすることです。
- DATA ノードが含まれていない場合は以下のとおりです。
  - 1 つの DM (分散キャッシュ・クラスタ・データ・サーバ、スタンドアロン InterSystems IRIS インスタンス、ネームスペース・レベルのシャード・クラスタ内のシャード・マスタ・データ・サーバ)。
  - ゼロ個以上のAM (分散キャッシュ・クラスタ・アプリケーション・サーバ)。
  - ゼロ個以上の DS (ネームスペース・レベルのシャード・クラスタ内のシャード・データ・サーバ)。
  - ゼロ個以上の QS (ネームスペース・レベルのシャード・クラスタ内のシャード・クエリ・サーバ、対応する DS ノードなしでは導入できない)
- ゼロ個以上の WS (Web サーバ)。
- ゼロ個以上の LB (ロード・バランサ)。
- ゼロ個以上の VM (仮想マシン・ノード)。
- ゼロ個以上の CN (コンテナ・ノード)。
- ゼロ個または 1 個の BH (要塞ホスト)。
- ゼロ個の AR (アービター・ノードはミラーリングされた構成の場合のみ)。

これらの一部のノード・タイプ間の関係を以下の例に図示します。

図 4-1: ICM のミラーリングされていないトポロジ





4.4.3 ミラーリングされた構成の要件

ミラーリングされたクラスタは、以下で構成されます。

- DATA ノード (ノード・レベルのシャード・クラスタ内のデータ・ノード) が含まれる場合は以下のとおりです。
  - 既定でまたは明示的に、**MirrorMap** の値と一致する個数の DATA。(″ミラーリングの規則″ を参照)。
  - ゼロ個以上の COMPUTE (ノード・レベルのシャード・クラスタ内の計算ノード)。ベスト・プラクティスは、DATA ノード・ミラーごとに少なくとも 1 つの COMPUTE ノードを含め、各 DATA ノード・ミラーの COMPUTE ノードの数を同じにすることです。
- DATA ノードが含まれていない場合は以下のとおりです。
  - ネームスペース・レベルのシャード・クラスタ内のミラーリングされたシャード・マスタ・データ・サーバ、分散キャッシュ・クラスタ内のデータ・サーバ、またはスタンドアロンの InterSystems IRIS インスタンスとしての 2 つの DM。DR 非同期が ″ミラーリングの規則″ の説明のとおり **MirrorMap** フィールドで指定されている場合は、2 つを超える DM。
  - ネームスペース・レベルのシャード・クラスタの場合は以下のとおりです。
    - 既定でまたは明示的に、**MirrorMap** の値と一致する個数の DS (シャード・データ・サーバ)。(″ミラーリングの規則″ を参照。)
    - ゼロ個以上の QS (シャード・クエリ・サーバ)。(前述の COMPUTE ノードの説明を参照。)
  - 分散キャッシュ・クラスタ内のアプリケーション・サーバとしてのゼロ個以上の AM。
- ゼロ個または 1 個の AR (アービター・ノードはオプションですが、ミラーリングされた構成の場合は推奨されます)。
- ゼロ個以上の WS (Web サーバ)。

- ゼロ個以上の LB (ロード・バランサ)。
- ゼロ個以上の VM (仮想マシン・ノード)。
- ゼロ個以上の CN (コンテナ・ノード)。
- ゼロ個または 1 個の BH (要塞ホスト)。

ミラーリングには、以下のフィールドを指定する必要があります。

- ミラーリングを有効にするには、**defaults.json** ファイルのキー **Mirror** を true に設定します。

  ```
  "Mirror": "true"
  ```

- DATA、DS、または DM ミラーに DR 非同期を含めるには、定義ファイルに **MirrorMap** フィールドを含めて、最初の 2 個以外が DR 非同期メンバであることを指定する必要があります。**MirrorMap** の値は、常に **primary,backup** で始まる必要があります。以下に例を示します。

  ```
  "Role": "DM",
  "Count": "5",
  "MirrorMap": "primary,backup,async,async,async",
  ...
  ```

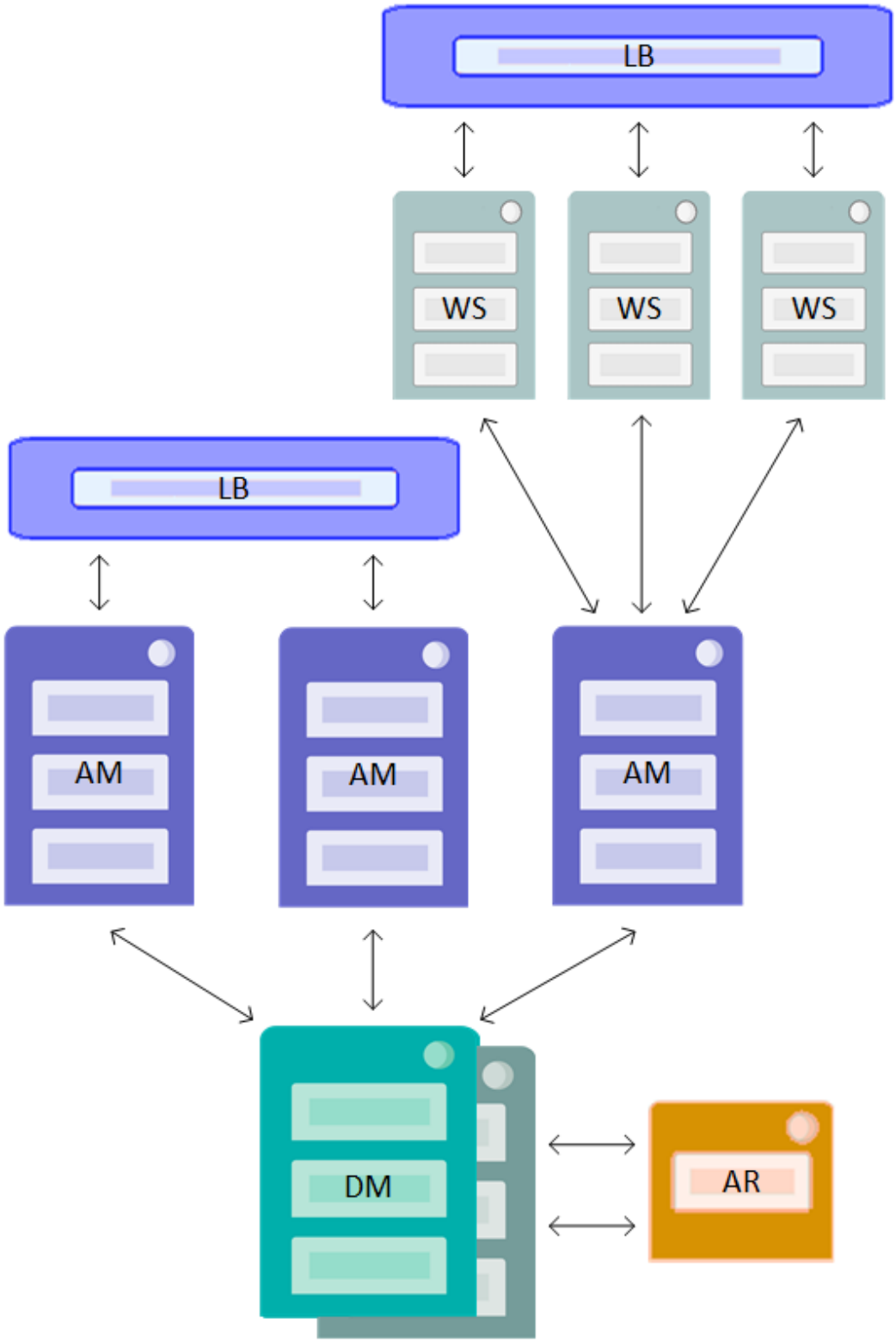
  MirrorMap の値と定義されている DATA、DS、または DM ノードの数との関係の詳細は、"ミラーリングの規則" を参照してください。**MirrorMap** を **Zone** フィールドおよび **ZoneMap** フィールドと組み合わせて使用して、複数のゾーンにわたって非同期インスタンスを導入することができます。"複数ゾーンにわたる導入" を参照してください。

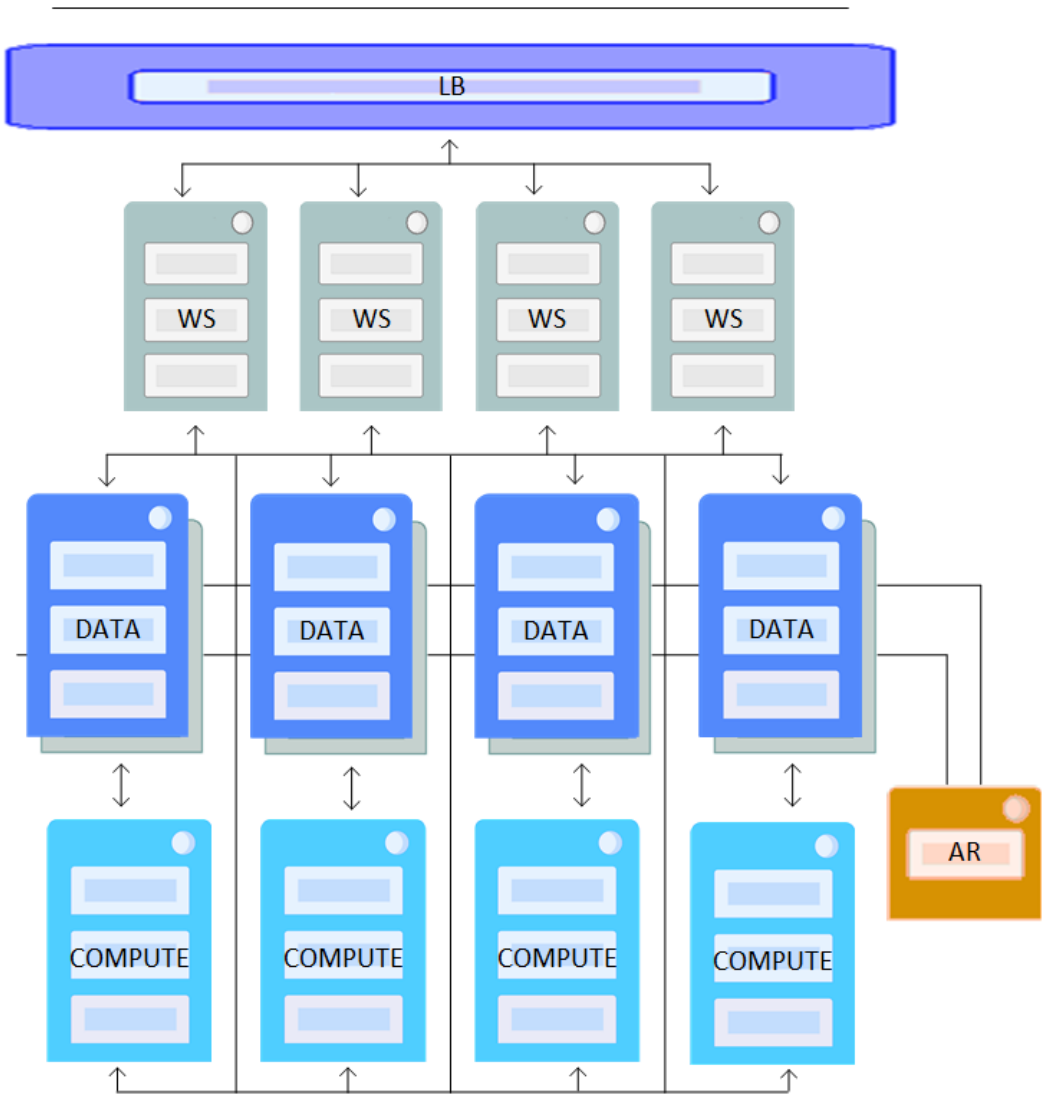
LB の自動導入 ("ロール LB：ロード・バランサ" を参照) は、AWS、GCP、Azure、および Tencent の各プロバイダについてサポートされています。独自のロード・バランサを作成する場合、組み込む IP アドレスのプールは、構成およびアプリケーションの要件に応じて、DATA ノード、COMPUTE ノード、AM ノード、または WS ノードのものです。

注釈　AM または WS ノードあるいはロード・バランサ (LB ノード) なしで導入されるミラーリングされた DM ノードには、フェイルオーバー後にアプリケーション接続をリダイレクトするための適切な仕組みが必要です。詳細は、"高可用性ガイド" の "ミラーリング" の章にある "フェイルオーバーまたは災害復旧後のアプリケーション接続のリダイレクト" を参照してください。

これらの一部のノード・タイプ間の関係を以下の例に図示します。

図 4-2: ICM のミラーリングされたトポロジ





4.5 ICM によってマウントされるストレージ・ボリューム

ICM は、InterSystems IRIS コンテナを導入する各ノードで、永続的な %SYS 機能 (″コンテナ内での InterSystems IRIS の実行″ の ″永続インスタンス・データを保存するための永続的な %SYS″ を参照) を使用して、InterSystems IRIS による永続データ・ストレージ用の 4 つのボリュームのフォーマット、パーティション分割、およびマウントを行います。これらのボリュームは、フィールド DataDeviceName (データ・ボリュームの場合)、WIJDeviceName (WIJ ディレクトリを含むボリュームの場合)、および Journal1DeviceName と Journal2DeviceName (プライマリ・ジャーナル・ディレクトリと代替ジャーナル・ディレクトリの場合) によって指定されたファイル名で、ホスト・ノード上の /dev/ に別個のデバイス・ファイルとしてマウントされます。これらのボリュームのサイズを指定するには、DataVolumeSize、WIJVolumeSize、Journal1VolumeSize、および Journal2VolumeSize の各パラメータ (″一般パラメータ″ を参照) を使用します。

タイプ PreExisting 以外のすべてのプロバイダについて、ICM は、以下のテーブルに示すように、デバイス名の妥当な既定値を割り当てようとします。ただし、これらの値はプラットフォームと OS に固有である場合が多く、defaults.json ファイルでオーバーライドする必要が生じることがあります (PreExisting の導入については、付録 “既存のクラスタへの導入” の ″ストレージ・ボリューム″ を参照してください)。

パラメータ	永続ボリュームの用途に対応するデバイス名	AWS	GCP	Azure	Tencent	vSphere
DataDeviceName	データベース	xvdd	sdc	sdd	vdc	sdc
WIJDeviceName	WIJ ディレクトリ	xvde	sdd	sde	vdd	sdd
Journal1DeviceName	プライマリ・ジャーナル・ディレクトリ	xvdf	sde	sdf	vde	sde
Journal2DeviceName	代替ジャーナル・ディレクトリ	xvdg	sdf	sdg	vdf	sdf

ICM は、以下の表に示すフィールドに従って、デバイスを InterSystems IRIS コンテナ内でマウントします。

パラメータ	既定値
DataMountPoint	/irissys/data
WIJMountPoint	/irissys/wij
Journal1MountPoint	/irissys/journal1
Journal2MountPoint	/irissys/journal2

このように配置することで、"コンテナ内でのインターシステムズ製品の実行" の "コンテナ化された InterSystems IRIS のファイル・システムの分離" で説明しているように、InterSystems IRIS で使用するストレージ用に別個のファイル・システムを使用して、パフォーマンスと復元可能性をサポートするうえで推奨されるベスト・プラクティスに、既定値をそのまま使用するだけで、容易に準拠することができます。

マシン・イメージに使用可能なマウント・ポイントが既にある場合、特別なデバイス名 existing をデバイス名パラメータの値として指定することで、ボリュームの割り当てをスキップして、対応するマウントポイント・パラメータで指定したディレクトリを使用するよう ICM に指示することができます。例えば、DataDeviceName の値が existing で、DataMountPoint の値が /mnt/data の場合、ICM は /mnt/data を InterSystems IRIS インスタンスのデータ・ボリュームとしてマウントします。マウント・ポイント・パラメータで指定したディレクトリが存在しない場合、データ・ボリュームはマウントされず、プロビジョニング時にエラーが表示されます。既存のディレクトリは、ユーザ **irisowner** (UID 51773) が書き込み可能である必要があります。"コンテナ内でのインターシステムズ製品の実行." の "InterSystems IRIS コンテナのセキュリティ" を参照してください (これは、プロバイダ PreExisting の既定のデバイス名および動作です)。

4.6 ICM の InterSystems IRIS ライセンス

コンテナに導入された InterSystems IRIS インスタンスには、コンテナ化されていないインスタンスと同様にライセンスが必要です。一般的な InterSystems IRIS ライセンスの要素と手順は、"システム管理ガイド" の "ライセンス" の章で説明されています。

ライセンス・キーを InterSystems IRIS コンテナ・イメージに含めることはできませんが、コンテナを作成して起動した後に追加する必要があります。ICM では、これは以下のように対処されています。

- 必要なライセンス・キーが、defaults.json file の LicenseDir フィールドで指定されている ICM コンテナ内のディレクトリまたはマウント・ボリューム (例 : /Samples/License) でステージングされます。

- ステージング・ディレクトリ内のライセンス・キーの 1 つが、definitions.json ファイル内のノード・タイプ DATA、COMPUTE、DM、AM、DS、および QS の各定義内の LicenseKey フィールドによって指定されます。以下に例を示します。

```
"Role": "DM",
"LicenseKey": "ubuntu-sharding-iris.key”,
"InstanceType": "m4.xlarge",
```

- ICM は、DATA ノード 1 または DM ノードにライセンス・サーバを構成します。このノードは、導入時に、指定されたライセンスを InterSystems IRIS ノード (自身も含む) に提供します。

重要

LicenseDir フィールドで指定されたディレクトリにステージングされ、LicenseKey フィールドで指定されたすべてのファイルは、.key 接尾語が付加された有効な InterSystems IRIS ライセンス・キー・ファイルであることが必要です。

関与する構成に関係なく、InterSystems IRIS が導入されるすべてのノードに、シャーディング対応の InterSystems IRIS ライセンスが必要です。

AR、LB、WS、VM、および CN ノードの場合、ライセンスは不要です。LicenseKey フィールドがこれらいずれかの定義に含まれている場合、このフィールドは無視されます。

ライセンスは SAM ノードではオプションですが、指定されている場合は使用されます。

4.7 ICM セキュリティ

ICM に組み込まれているセキュリティ手段について、以下のセクションで説明します。

- ホスト・ノードの通信
- Docker
- Weave Net
- InterSystems IRIS
- プライベート・ネットワーク

ここで説明するセキュリティのために必要なファイルの指定に使用される ICM フィールドの詳細は、"セキュリティ関連のパラメータ" を参照してください。

4.7.1 ホスト・ノードの通信

ホスト・ノードは、コンテナが導入されるホスト・マシンです。これは仮想マシンまたは物理マシンで、クラウド内またはオンプレミスで実行されます。

ICM は、SSH を使用してホスト・ノードにログインし、それらに対してリモートでコマンドを実行します。また、SCP を使用して、ICM コンテナとホスト・ノードの間でファイルをコピーします。このセキュリティ保護された通信を有効にするには、SSH 公開/秘密鍵ペアを提供し、これらの鍵を defaults.json ファイルで SSHPublicKey および SSHPrivateKey として指定する必要があります。構成フェーズ中に、ICM は各ホスト・ノードでパスワード・ログインを無効にし、秘密鍵をノードにコピーし、ポート 22 を開いて、対応する公開鍵を持つクライアントが SSH および SCP を使用してノードに接続できるようにします。

ホスト・マシン上で開かれるその他のポートについては、以降のセクションで説明します。

4.7.2 Docker

プロビジョニング時に、ICM は GPG 指紋を使用して Docker 公式 Web サイトから Docker の特定バージョンをダウンロードし、インストールします。ICM はその後、提供された TLS 証明書 (既定値ファイルの TLSKeyDir フィールドで指定されたディレクトリに配置されている) をホスト・マシンにコピーし、TLS を有効にして Docker デーモンを起動して、ポート 2376 を開きます。この時点で、対応する証明書を持つクライアントは、ホスト・マシンに Docker コマンドを発行できます。

4.7.3 Weave Net

プロビジョニング時に、Weave ネットワークに接続する各マシンからパスワード (ユーザが提供する) を要求するオプションとトラフィック暗号化オプションで、Weave Net を ICM は起動します。これらのオプションを有効にするには、defaults.json ファイルで WeavePassword を null 以外の任意の値に設定します。

4.7.4 InterSystems IRIS

InterSystems IRIS のセキュリティの包括的な概要については、"インターシステムズのセキュリティについて" を参照してください。

4.7.4.1 セキュリティ・レベル

InterSystems IRIS イメージが標準のセキュリティ (最小またはロック・ダウンではない) でインストールされたことを ICM は前提とします。

4.7.4.2 事前定義アカウントのパスワード

InterSystems IRIS インスタンスを保護するために、事前定義アカウントの既定のパスワードは ICM で変更する必要があります。ICM が InterSystems IRIS コンテナを初めて実行するときに、NULL でないロールを持つ有効なアカウントすべてのパスワードが、ユーザ提供のパスワードに変更されます。InterSystems IRIS パスワードが定義ファイル内で見えないように、また -iscPassword オプションを使用してコマンド行の履歴に表示されないようにする場合は、両方とも省略できます。ICM はインタラクティブにパスワードの入力を求め、入力内容はマスクされます。パスワードは保持されるので、InterSystems IRIS コンテナを再起動またはアップグレードしても変更されません。

4.7.4.3 JDBC

ICM は、既定値ファイルの TLSKeyDir フィールドで指定されたディレクトリに配置されているファイルを使用して、InterSystems IRIS への JDBC 接続を (InterSystems IRIS の要件に従って) TLS モードで開きます。

4.7.4.4 ミラーリング

ICM は、既定値ファイルの TLSKeyDir フィールドで指定されたディレクトリに配置されているファイルを使用し、TLS を有効にしてミラーを作成します ("高可用性ガイド" の "ミラーリング" の章を参照)。フェイルオーバー・メンバは、TLS が有効な場合のみミラーに参加できます。

4.7.4.5 インターシステムズの Web ゲートウェイ

ICM は、既定値ファイルの TLSKeyDir フィールドで指定されたディレクトリに配置されているファイルを使用し、TLS を使用して DM ノードおよび AM ノードと通信するように WS ノードを構成します。

4.7.4.6 InterSystems ECP

ICM は、すべての InterSystems IRIS ノードが ECP 接続に TLS を使用するように構成します。これには、分散キャッシュ・クラスタ・ノードとシャード・クラスタ・ノードの間の接続も含まれます。

4.7.4.7 セキュリティの一元化

LDAP サーバを使用してシャード・クラスタまたはその他の ICM 導入環境のノード全体で一元的なセキュリティを実装することをお勧めします。InterSystems IRIS での LDAP の使用についての詳細は、"LADP ガイド" を参照してください。

4.7.5 プライベート・ネットワーク

ICM は、インターネットからアクセスできない既存のプライベート・ネットワークで導入を行うことができます (必要なアクセスが構成されている場合)。また、導入を行うプライベート・ネットワークを作成し、要塞ホストを介した独自のアクセスを構成することもできます。プライベート・ネットワークの使用の詳細は、"プライベート・ネットワークへの導入" を参照してください。

4.8 カスタマイズされた InterSystems IRIS 構成を使用した導入

いずれの InterSystems IRIS インスタンスも、InterSystems IRIS コンテナ内で実行されるものを含め、インストール・ディレクトリ内の iris.cpf という名前のファイルを使用してインストールされます。このファイルには、ほとんどの構成設定が含まれています。インスタンスでは、起動時にこの構成パラメータ・ファイル、つまり CPF が読み取られ、そこに含まれる設定値が取得されます。設定が変更されると、CPF は自動的に更新されます。CPF の使用と内容については、"構成パラメータ・ファイル・リファレンス" で詳しく説明しています。

ただし、同じイメージから複数のインスタンスを異なる構成設定で導入する場合があります。これを実行するには、**ISC_CPF_MERGE_FILE** 環境変数を使用します。これにより、インスタンスの CPF にマージされる 1 つ以上の設定を含む個別のファイルを指定できます。構成マージ機能を使用して、複数のインスタンスを同じソースからの異なる CPF で導入できます。構成マージ機能の詳細は、"構成マージを使用した InterSystems IRIS の自動構成" を参照してください。

この機能は、**iris** コンテナまたはコンテナレス・インストールに適用する構成マージ・ファイルを指定する UserCPF プロパティを使用して ICM で InterSystems IRIS を導入する場合に利用できます。例えば、インターシステムズ提供の InterSystems IRIS イメージに含まれる CPF の [config] セクションには、以下のような既定の共有メモリ・ヒープ構成 ("システム管理ガイド" の "InterSystems IRIS の構成" の章にある "共有メモリ・ヒープ (gmheap) の構成" を参照) が含まれています。

```
[config]
LibPath=
MaxServerConn=1
MaxServers=2
...
gmheap=37568
...
```

導入環境内のすべての InterSystems IRIS インスタンスについて共有メモリ・ヒープのサイズを 2 倍にするには、以下の内容を含む merge.cpf というファイルを ICM コンテナ内に作成します。

```
[config]
gmheap=75136
```

その後、以下のように、UserCPF フィールドを使用して defaults.json でこのマージ・ファイルを指定します。

```
"UserCPF": "/Samples/mergefiles/merge.cpf"
```

これにより、導入された各 InterSystems IRIS インスタンスが起動される前に、各インスタンスの CPF が新しい共有メモリ・ヒープ・サイズで更新されます。

定義ファイルでこのフィールドを使用して、特定のノード・タイプのみにマージ・ファイルを適用することもできます。例えば、分散キャッシュ・クラスタ内の DM ノードのみで共有メモリ・ヒープのサイズを 2 倍にし、同時に AM ノードで ECP の

[リカバリまでの待機時間] 設定を既定の 1200 秒から 1800 秒に変更するには、以下の内容を含む merge2.cpf という別のファイルを作成します。

```
[ECP]
ClientReconnectDuration=1800
```

その後、以下のような definitions.json ファイルを使用します。

```
[
    {
        "Role": "DM",
        "Count": "1",
        "UserCPF": "/Samples/mergefiles/merge.cpf"
    },
    {
        "Role": "AM",
        "Count": "3",
        "StartCount": "2",
        "UserCPF": "/Samples/mergefiles/merge2.cpf",
        "LoadBalancer": "true"
    }
]
```

この場合、DM ノードでは共有メモリ・ヒープ・サイズが 2 倍になりますが、AM ノードでは 2 倍になりません。また、AM ノードでは ECP 設定が変更されますが、DM ノードでは変更されません。

4.9 複数ゾーンにわたる導入

クラウド・プロバイダでは通常、指定されたリージョン内の複数のゾーンにまたがって仮想ネットワークを利用することができます。導入環境によっては、これを利用して、それぞれのノードを異なるゾーンに導入できます。例えば、各データ・ノードにフェイルオーバー・ペアと DR 非同期が含まれる、ミラーリングされたシャード・クラスタを導入する場合 ("ミラーリングされた構成の要件" を参照)、フェイルオーバー・ペアと DR 非同期を 2 つの異なるゾーンに導入することで、物理的な DR 非同期をリモート・データ・センタに配置する環境と同等のクラウドを実現できます。

AWS、GCP、Azure、および Tencent への導入時に複数のゾーンを指定するには、既定値ファイルの Zone フィールドにコンマ区切りのゾーン・リストを入力します。以下は AWS の場合の例です。

```
{
    "Provider": "AWS",
    ...
    "Region": "us-west-1",
    "Zone": "us-west-1b,us-west-1c"
}
```

GCP の場合 :

```
    "Provider": "GCP",
    ...
    "Region": "us-east1",
    "Zone": "us-east1-b,us-east1-c"
}
```

Azure の場合 :

```
    "Provider": "Azure",
    ...
    "Region": "Central US",
    "Zone": "1,2"
```

Tencent の場合 :

```
    "Provider": "Tencent",
    ...
    "Region": "na-siliconvalley",
    "Zone": "na-siliconvalley-1,na-siliconvalley-2"
```

指定したゾーンは、ラウンドロビン方式でノードに割り当てられます。例えば、AWS の例を使用して、4 つのミラーリングされていない DATA ノードをプロビジョニングする場合、最初と 3 つ目が us-west-1b でプロビジョニングされ、2 つ目と 4 つ目が us-west-1c でプロビジョニングされます。

ただし、ミラーリングされた構成の場合、ラウンドロビン方式の分散が、望ましくない結果につながる可能性があります。例えば、前にある Zone の指定により、ミラーリングされた DATA、DM または DS ノードのプライマリ・メンバとバックアップ・メンバが異なるゾーンに配置されたときに、これが、メンバ間の遅延を大きくするため、使用しているアプリケーションには適していない場合があります (″高可用性ガイド″ の ″ネットワーク遅延に関する考慮事項″ を参照)。どのノードをどのゾーンに配置するかを選択するには、definitions.json ファイルのノード定義に ZoneMaps フィールドを追加して、Zone フィールドで指定される特定のゾーンに、単一ノード、または複数ノードのゾーン配置のパターンを指定できます。これは、ミラーリングされたデータ・サーバが設定されている分散キャッシュ・クラスタに関する以下の指定で示されています。

defaults.json

```
"Mirror": "True"
"Region": "us-west-1",
"Zone": "us-west-1a,us-west-1b,us-west-1c"
```

definitions.json

```
"Role": "DM",
"Count": "4″,
"MirrorMap": "primary,backup,async,async",
"ZoneMap": "0,0,1,2",
...
"Role": "AM",
"Count": "3″,
"MirrorMap": "primary,backup,async,async",
"ZoneMap": "0,1,2",
...
"Role": "AR",
...
```

この指定によって、プライマリおよびバックアップのミラー・メンバが us-west-1a に配置され、1 つのアプリケーション・サーバが各ゾーンに配置されます。一方、非同期は必要に応じて可用性を最大にするためにフェイルオーバー・ペアとは異なるゾーンに配置され、最初の非同期が us-west-1b、2 つ目の非同期が us-west-1c に配置されます。アービター・ノードでは、ZoneMap フィールドをフェイルオーバー・ペアと一緒に us-west-1a に配置する必要はありません。ラウンドロビン方式の分散がこれを行います。

このアプローチを以下のように、各データ・ノード・ミラーにフェイルオーバー・ペアと DR 非同期が含まれる、ミラーリングされたシャード・クラスタで使用することもできます。

defaults.json

```
"Mirror": "True"
"Region": "us-west-1",
"Zone": "us-west-1a,us-west-1b,us-west-1c"
```

definitions.json :

```
"Role": "DATA",
"Count": "12",
"MirrorMap": "primary,backup,async",
"ZoneMap": "0,0,1",
...
"Role": "COMPUTE",
"Count": "8",
"ZoneMap": "0",
...
"Role": "AR",
"ZoneMap": "2",
...
```

この指定によって、4 つのデータ・ノード・ミラーそれぞれのフェイルオーバー・ペアと 8 つの計算ノードが us-west-1a に、各データ・ノード・ミラーの DR 非同期が us-west-1b に、アービターが us-west-1c に配置されます。

4.10 複数のリージョンまたはプロバイダにわたる導入

ICM は複数のクラウド・プロバイダ・リージョンにわたって導入できます。例えば、DR 非同期ミラー・メンバを、フェイルオーバー・メンバとは異なるリージョンに配置できます。マルチリージョン導入の手順はプロバイダによって異なり、これについては以下のセクションで説明します。3 つめのセクションで説明する手順は、複数のプロバイダにわたる導入にも使用できます。

- GCP での複数のリージョンにわたる導入
- Azure での複数のリージョンにわたる導入
- AWS と Tencent での複数のリージョンにわたる導入

重要　ミラーのフェイルオーバー・メンバを異なるリージョンまたは異なるプラットフォームに導入することもできますが、リージョン間やプラットフォーム間では一般的にネットワーク遅延が大きいことが原因でミラー処理の問題が発生することから、このような方法はお勧めしません。ミラーにおける遅延に関する考慮事項の詳細は、"高可用性ガイド" の "ミラーリング" の章にある "ネットワーク遅延に関する考慮事項" を参照してください。

注釈　複数のリージョンにわたる導入とプライベート・ネットワークへの導入 ("プライベート・ネットワークへの導入" を参照) は、このリリースでは互換性がありません。

4.10.1 GCP での複数のリージョンにわたる導入

GCP で複数のリージョンにわたる導入を行うには、既定値ファイルの Region フィールドで、以下のように目的のリージョンをコンマ区切りリストとして指定します。

```
{
    "Provider": "GCP",
    "Label": "Sample",
    "Tag": "multi",
    "Region": "us-east1,us-west1",
    "Zone": "us-east1-b,us-west1-a",
    ...
}
```

既定では、各定義内のノードに、ラウンドロビン方式でリージョンが割り当てられます。例えば、上の defaults.json と下の definitions.json に示すフィールドを使用して導入するとします。

```
[
  {
    "Role": "DATA",
    "Count": "2"
  },
  {
    "Role": "AR",
    "Count": "1",
    "DockerImage": "intersystems/arbiter:latest-em"
  }
]
```

この場合、icm inventory コマンドの出力は次のようになります。

```
$ icm inventory
Machine               IP Address    DNS Name              Region    Zone
-------               ----------    --------              ------    ----
Acme-AR-multi-0001    35.179.173.90 acmear1.google.com    us-east1     b
Acme-DATA-multi-0001- 35.237.131.39 acmedata1.google.com  us-east1     b
Acme-DATA-multi-0002+ 35.233.223.64 acmedata2.google.com  us-west1     a
```

ノードが導入されるリージョンを制御するために、RegionMap フィールドを使用して、定義されたノードを指定のリージョンにマップすることができます。RegionMap をノード定義に含めたら、ZoneMap (前のセクション ″複数ゾーンにわたる導入″ を参照) も含めて、ノード (複数可) を目的のゾーン (複数可) にマップする必要があります。例えば、フェイルオーバー・ペアと DR 非同期がアービターと共に含まれるミラーを導入するとします。フェイルオーバー・ペアは同じリージョンの異なるゾーンに、非同期とアービターは異なるリージョンの異なるゾーンに導入します。使用するファイルおよび icm inventory コマンドと icm ps コマンドの出力を、以下に示します。

defaults.json

```
{
    "Provider": "GCP",
    "Label": "Sample",
    "Tag": "multi",
    "Region": "us-east1,us-west1",
    "Zone": "us-east1-a,us-east1-b,us-west1-a,us-west1-b",
    ...
}
```

definitions.json

```
[
  {
    "Role": "DATA",
    "Count": "3",
    "MirrorMap": "primary,backup,async",
    "RegionMap": "1,1,2",
    "ZoneMap": "1,2,3",
  },
  {
    "Role": "AR",
    "Count": "1",
    "DockerImage": "intersystems/arbiter:latest-em",
    "RegionMap": "2",
    "ZoneMap": "4"
  }
]
```

icm inventory

```
Machine               IP Address    DNS Name              Region    Zone
-------               ----------    --------              -------   ----
Acme-AR-multi-0001    35.179.173.90 acmear1.google.com    us-west1     b
Acme-DATA-multi-0001+ 35.237.131.39 acmedata1.google.com  us-east1     b
Acme-DATA-multi-0002- 35.233.223.64 acmedata2.google.com  us-east1     a
Acme-DATA-multi-0003  35.166.127.82 acmedata3.google.com  us-west1     a
```

icm ps

```
Machine              IP Address    Container Status Health  Mirror    Image
-------              ----------    --------- ------ ------  ------    -----
Acme-AR-multi-0001   35.179.173.90 arbiter   Up     healthy           intersystems/arbiter:latest-em
Acme-DATA-multi-0001 35.237.131.39 iris      Up     healthy PRIMARY   intersystems/iris:latest-em
Acme-DATA-multi-0002 35.233.223.64 iris      Up     healthy BACKUP    intersystems/iris:latest-em
Acme-DATA-multi-0003 35.166.127.82 iris      Up     healthy CONNECTED intersystems/iris:latest-em
```

Network フィールド ("Google Cloud Platform (GCP) パラメータ" を参照) を使用して、マルチリージョン導入で使用する既存のネットワークを指定するには、GCP Subnet フィールドを使用して、Region フィールドで指定されたリージョンごとに一意のサブネットを指定することも必要です。例えば、ここで示したリージョン us-west1 および us-east1 への導入の場合は、既定値ファイルに以下を含める必要があります。

```
"Network": "acme-network",
"Subnet": "acme-subnet-data-east,acme-subnet-data-west"
```

注釈　GCP マルチリージョン導入にロード・バランサ (LB ノード) を含めることはできません。ロード・バランサは GCP 上の単一ノードに制限されているためです。

4.10.2 Azure での複数のリージョンにわたる導入

Azure で複数のリージョンにわたる導入を行うには、既定値ファイルの Location フィールドで、以下のように目的のリージョンをコンマ区切りリストとして指定します。

```
{
    "Provider": "Azure",
    "Label": "Sample",
    "Tag": "multi",
    "Location": "East US,Central US",
    ...
}
```

既定では、各定義内のノードに、ラウンドロビン方式で場所が割り当てられます。例えば、上の defaults.json と下の definitions.json に示すフィールドを使用して導入するとします。

```
[
  {
    "Role": "DATA",
    "Count": "2"
  },
  {
    "Role": "AR",
    "Count": "1",
    "DockerImage": "intersystems/arbiter:latest-em"
  }
]
```

この場合、icm inventory コマンドの出力は次のようになります。

```
$ icm inventory
Machine               IP Address    DNS Name             Region    Zone
-------               ----------    --------             ------    ----
Acme-AR-multi-0001    35.179.173.90 acmear1.azure.com    East US    1
Acme-DATA-multi-0001- 35.237.131.39 acmedata1.azure.com  East US    1
Acme-DATA-multi-0001+ 35.233.223.64 acmedata2.azure.com  Central US 1
```

ノードが導入されるリージョンを制御するために、LocationMap フィールドを使用して、定義されたノードを指定のリージョンにマップすることができます。LocationMap をノード定義に含めたら、ZoneMap (前のセクション "複数ゾーンにわたる導入" を参照) も含めて、ノード (複数可) を目的のゾーン (複数可) にマップする必要があります。例えば、フェイルオーバー・ペアと DR 非同期がアービターと共に含まれるミラーを導入するとします。フェイルオーバー・ペアは同じリージョンの異なるゾーンに、非同期とアービターは異なるリージョンの異なるゾーンに導入します。使用するファイルおよび icm inventory コマンドと icm ps コマンドの出力を、以下に示します。(Azure のゾーンはすべてのリージョンで同じ整数によっ

て識別されるため、ZoneMap は LocationMap でどのリージョンが指定されていてもその中の目的のゾーンのみを識別します。)

defaults.json

```
{
    "Provider": "Azure",
    "Label": "Sample",
    "Tag": "multi",
    "Location": "East US,Central US",
    "Zone": "1,2",
    ...
}
```

definitions.json

```
[
  {
    "Role": "DATA",
    "Count": "3",
    "MirrorMap": "primary,backup,async",
    "LocationMap": "1,1,2",
    "ZoneMap": "1,2,1"
  },
  {
    "Role": "AR",
    "Count": "1",
    "DockerImage": "intersystems/arbiter:latest-em",
    "RegionMap": "2",
    "ZoneMap": "2"
  }
]
```

icm inventory

```
Machine               IP Address    DNS Name             Region     Zone
-------               ----------    --------             ------     ----
Acme-AR-multi-0001    35.179.173.90 acmear1.azure.com    Central US 2
Acme-DATA-multi-0001+ 35.237.131.39 acmedata1.azure.com  East US    1
Acme-DATA-multi-0002- 35.233.223.64 acmedata2.azure.com  East US    2
Acme-DATA-multi-0003  35.166.127.82 acmedata3.google.com Central US 1
```

icm ps

```
Machine              IP Address    Container Status Health  Mirror    Image
-------              ----------    --------- ------ ------  ------    -----
Acme-AR-multi-0001   35.179.173.90 arbiter   Up     healthy           intersystems/arbiter:latest-em
Acme-DATA-multi-0001 35.237.131.39 iris      Up     healthy PRIMARY   intersystems/iris:latest-em
Acme-DATA-multi-0002 35.233.223.64 iris      Up     healthy BACKUP    intersystems/iris:latest-em
Acme-DATA-multi-0003 35.166.127.82 iris      Up     healthy CONNECTED intersystems/iris:latest-em
```

Azure マルチリージョン導入環境で既存の仮想ネットワークを使用する場合は、既定値ファイルに ResourceGroupName および VirtualNetwork フィールドを含めて ("Microsoft Azure (Azure) パラメータ" を参照)、Location フィールドで指定されたリージョンごとにネットワークを指定する必要があります。以下に例を示します。

```
{
    "Provider": "Azure",
    "Label": "Sample",
    "Tag": "multi",
    "Location": "East US,Central US",
    "Zone": "1,2",
    "ResourceGroupName": "sample-resource-group",
    "VirtualNetworkName": "sample-vnet-east,sample-vnet-central"
    ...
}
```

指定したネットワークは、重複しないアドレス空間を持つ必要があります。付随する定義ファイルでは、各定義に、Location フィールドで指定されたリージョンごとに一意のサブネットを指定する Subnetname フィールドを含める必要があります。例えば、このセクションで説明するミラーリングされた導入で、既定値ファイルに ResourceGroupName および

VirtualNetwork フィールドが含まれる場合、定義ファイルは次のようになります。AR 定義は Central US リージョンにのみ導入するため、この定義で必要なサブネットは 1 つだけです。

```
[
  {
    "Role": "DATA",
    "Count": "3",
    "MirrorMap": "primary,backup,async",
    "RegionMap": "1,1,2",
    "ZoneMap": "1,2,1",
    "SubnetName": "acme-subnet-data-east,acme-subnet-data-central"
  },
  {
    "Role": "AR",
    "Count": "1",
    "DockerImage": "intersystems/arbiter:latest-em",
    "RegionMap": "2",
    "ZoneMap": "2",
    "SubnetName": "acme-subnet-arbiter-central"
  }
]
```

注釈　Azure マルチリージョン導入環境にロード・バランサ (LB ノード) を含めることはできません。ロード・バランサは Azure 上の単一ノードに制限されているためです。

4.10.3 AWS と Tencent での複数のリージョンにわたる導入

AWS と Tencent で複数のリージョンにわたって導入するには、ICM はまず必要なインフラストラクチャを別個のリージョンにプロビジョニングし、次に、そのインフラストラクチャをマージして、あたかも既存のインフラストラクチャであるかのようにそこにサービスを導入します。

この手順は、複数のプロバイダにわたる導入にも使用できます。この説明では、"リージョンまたはプロバイダ" を示す場合に "リージョン" を使用し、マルチプロバイダと AWS/Tencent マルチリージョンとの違いは必要に応じて記載します。

マージされたマルチリージョン導入を作成する手順は、以下のステップで構成されます。

1. 別個の ICM セッションでリージョンごとにインフラストラクチャをプロビジョニングします。
2. icm merge コマンドを使用して、マルチリージョン・インフラストラクチャをマージします。
3. マージされた definitions.json ファイルを確認し、必要に応じて順序変更や更新を行います。
4. icm provision コマンドを使用して、マージされたインフラストラクチャを再プロビジョニングします。
5. icm run コマンドを使用し、PreExisting 導入として、マージされたインフラストラクチャにサービスを導入します。
6. インフラストラクチャをプロビジョニング解除する場合は、元のセッション・ディレクトリで別途 icm unprovision コマンドを発行します。

4.10.3.1 インフラストラクチャのプロビジョニング

リージョン (Region フィールドで指定されたもの) ごとにインフラストラクチャをプロビジョニングするための別個のセッションを、同じ ICM コンテナ内の別個の作業ディレクトリで実行する必要があります。例えば、提供されている /Samples/AWS ディレクトリ (“ICM の使用” の章にある "導入の定義" を参照) を /Samples/AWS/us-east-1 および /Samples/AWS/us-west-1 にコピーすることから始めることができます。それぞれの既定値ファイルと定義ファイルで、最終的なマルチリージョン導入環境に合わせて、目的のリージョン、ノード定義、および機能を指定します。例えば、一方のリージョンにミラー・フェイルオーバー・ペアを導入し、もう一方のリージョンにミラーの DR 非同期メンバを導入する場合、適切なリージョンとゾーン、および "Mirror": "true" を既定値ファイルに含め、一方のリージョンの定義ファイルで 2 つの DM (フェイルオーバー・ペア用) を、他方の定義ファイルで 3 つ目の DM (非同期用) を、どちらかで 1 つの AR (アービター) ノードを定義します。リソースの競合を回避するために、マルチリージョン導入環境の既定値ファイルにはそれぞれ、一意の Label または Tag、あるいはその両方が必要です。マルチプロバイダ導入環境では、その必要はありません。この例を以下に示します。

注釈　Mirror が true に設定されている場合に DM ノードが 1 つしか定義されていないなど、指定された定義が単一リージョン導入環境のトポロジ要件を満たしていない場合は、/Samples/AWS/us-west-1/defaults.json に示すように、既定値ファイルに "SkipTopologyValidation": "true" を含めることによって、トポロジ検証を無効にします。

/Samples/AWS/us-east-1

defaults.json

```
{
    "Provider": "AWS",
    "Label": "Sample",
    "Tag": "east1",
    "Region": "us-east-1",
    "Zone": "us-east1-a,us-east1-b",
    "Mirror": "true",
    ...
}
```

definitions.json

```
[
  {
    "Role": "DM",
    "Count": "2",
    "ZoneMap": "1,2"
  }
]
```

/Samples/AWS/us-west-1/

defaults.json

```
{
    "Provider": "AWS",
    "Label": "Sample",
    "Tag": "west1",
    "Region": "us-west-1",
    "Zone": "us-west1-a,us-west1-b",
    "Mirror": "true",
    "SkipTopologyValidation": "true"},
    ...
}
```

definitions.json

```
[
  {
    "Role": "DM",
    "Count": "1",
    "ZoneMap": "1"
  }
  {
    "Role": "AR",
    "Count": "1",
    "ZoneMap": "2"
  }
]
```

各作業ディレクトリで icm provision コマンドを使用して、リージョンごとにインフラストラクチャをプロビジョニングします。各ディレクトリで実行された icm inventory コマンドの出力には、作業しているインフラストラクチャが表示されます。以下に例を示します。

/Samples/AWS/us-east-1

```
$ icm inventory
Machine             IP Address    DNS Name                          Region    Zone
-------             ----------    --------                          ------    ----
Acme-DM-east1-0001+ 54.214.230.24 ec2-54-214-230-24.amazonaws.com   us-east-1 a
Acme-DM-east1-0002- 54.129.103.67 ec2-54-129-103-67.amazonaws.com   us-east-1 b
```

/Samples/AWS/us-west-1

```
$ icm inventory
Machine             IP Address    DNS Name                          Region    Zone
-------             ----------    --------                          ------    ----
Acme-AR-west1-0001  54.181.212.79 ec2-54-181-212-79.amazonaws.com  us-west-1 b
Acme-DM-west1-0002  54.253.103.21 ec2-54-253-103-21.amazonaws.com  us-west-1 a
```

4.10.3.2 プロビジョニングされたインフラストラクチャのマージ

icm merge コマンドは、現在の作業ディレクトリおよび指定された追加のディレクトリ内の構成ファイルをスキャンして、指定の新しいディレクトリ内に、PreExisting 導入に使用できるマージされた構成ファイルを作成します。例えば、/Samples/AWS/us-east-1 および /Samples/AWS/us-west-1 内の定義ファイルと既定値ファイルをマージして /Samples/AWS/merge 内に新しいセットを作成するには、以下のコマンドを発行します。

```
$ cd /Samples/AWS/us-east-1
$ mkdir ../merge
$ icm merge -options ../us-west1 -localPath /Samples/AWS/merge
```

icm merge コマンドで、-options はローカル・ディレクトリとマージされるプロビジョニング・ディレクトリのコンマ区切りリストを指定し、--localPath はマージされる定義のマージ先ディレクトリを指定します。(ICM コマンド行に Docker 引数を含めることを可能にする -options オプションの詳細は、"カスタム・コンテナおよびサードパーティ・コンテナでの ICM の使用" を参照してください。)

4.10.3.3 マージされた定義ファイルの確認

新しい構成ファイルを調べると、マージされた既定値ファイルで Provider が PreExisting に変更されていることがわかります (以前の Provider フィールドや他のフィールドは定義ファイルに移動されています。これらは icm inventory コマンドによって表示されますが、それ以外の影響はありません)。必要に応じて、Label または Tag (あるいはその両方) を変更できます。

マージされた定義ファイル内の定義は、プロバイダ PreExisting で使用するように変換されています。付録 "既存のクラスタへの導入" の "PreExisting の定義ファイル" に記載されているように、PreExisting 導入の definitions.json ファイルには、ノードごとにエントリが 1 つずつ含まれます (ロールごとに 1 つのエントリがあり、そのロールのノード数を指定する Count フィールドが含まれるわけではありません)。それぞれのノードは、IP アドレスまたは完全修飾ドメイン名によって識別されます。IPAddress フィールドまたは DNSName フィールドに加えて、SSHUser フィールドがそれぞれの定義に含まれている必要があります ("既存のクラスタへの導入" の "SSH" に記載されているように、後者では、パスワードを使用しない sudo アクセス権を持つ非 root ユーザを指定します)。マージされたファイルでは、定義がリージョン別に (マルチプロバイダ導入環境ではプロバイダ別に) グループ化されています。ミラー・メンバの目的の配置を反映するように必要に応じて順序を変更し、適切なミラー・マップを定義する必要があります ("ミラーリングされた構成の要件" および "複数ゾーンにわたる導入" を参照)。確認が完了すると、この例の定義ファイルは以下のようになります。

```
[
    {
        "Role":"DM",
        "IPAddress":"54.214.230.24",
        "LicenseKey": "ubuntu-sharding-iris.key",
        "SSHUser": "icmuser",
        "MirrorMap": "primary,backup,async"
    },
    {
        "Role":"DM",
        "IPAddress":"54.129.103.67",
        "LicenseKey": "ubuntu-sharding-iris.key",
        "SSHUser": "icmuser",
        "MirrorMap": "primary,backup,async"
    },
    {
        "Role":"DM",
        "IPAddress":"54.253.103.21",
        "LicenseKey": "ubuntu-sharding-iris.key",
        "SSHUser": "icmuser",
        "MirrorMap": "primary,backup,async"
    },
    {
        "Role":"AR",
```

```
        "IPAddress":"54.181.212.79",
        "SSHUser": "icmuser",
        "StartCount": "4"
    }
]
```

4.10.3.4 マージされたインフラストラクチャの再プロビジョニング

新しいディレクトリ (この例では /Samples/AWS/merge) で icm provision コマンドを発行して、マージされたインフラストラクチャを再プロビジョニングします。icm inventory コマンドの出力には、マージされたインフラストラクチャが 1 つのリストとして表示されます。

```
$ icm inventory
Machine             IP Address     DNS Name                        Region    Zone
-------             ----------     --------                        ------    ----
Acme-DM-east1-0001+ 54.214.230.24 ec2-54-214-230-24.amazonaws.com us-east-1 a
Acme-DM-east1-0002- 54.129.103.67 ec2-54-129-103-67.amazonaws.com us-east-1 b
Acme-AR-west1-0001  54.181.212.79 ec2-54-181-212-79.amazonaws.com us-west-1 b
Acme-DM-west1-0002  54.253.103.21 ec2-54-253-103-21.amazonaws.com us-west-1 a
```

4.10.3.5 マージされたインフラストラクチャへのサービスの導入

いずれの導入環境でも行うように、icm run コマンドを使用して、マージされたインフラストラクチャにサービスを導入します。以下に例を示します。

```
$ icm run
...
-> Management Portal available at: http://112.97.196.104.google.com:52773/csp/sys/UtilHome.csp
$ icm ps
Machine             IP Address    Container Status Health  Mirror    Image
-------             ----------    --------- ------ ------  ------    -----
Acme-AR-multi-0001 35.179.173.90 arbiter   Up     healthy           intersystems/arbiter:latest-em
Acme-DM-multi-0001 35.237.131.39 iris      Up     healthy PRIMARY   intersystems/iris:latest-em
Acme-DM-multi-0002 35.233.223.64 iris      Up     healthy BACKUP    intersystems/iris:latest-em
Acme-DM-multi-0003 35.166.127.82 iris      Up     healthy CONNECTED intersystems/iris:latest-em
```

4.10.3.6 マージされたインフラストラクチャのプロビジョニング解除

マルチリージョン導入環境をプロビジョニング解除する時期が来たら、元の作業ディレクトリに戻って icm unprovision コマンドを発行した後、マージされた作業ディレクトリを削除します。この例では、以下のようにします。

```
$ cd /Samples/AWS/us-east-1
$ icm unprovision -force -cleanUp
...
...completed destroy of Acme-east1
$ cd /Samples/AWS/us-west-1
$ icm unprovision -force -cleanUp
...
...completed destroy of Acme-west1
$ rm -rf /Samples/AWS/merge
```

4.11 プライベート・ネットワークへの導入

ICM は、対象のロールに必要なポートおよびプロトコルのみを公開するように各ホスト・ノードのファイアウォールを構成します。例えば、ISCAgent ポートが公開されるのは、ミラーリングが有効であり、ロールが AR、DATA、DM、または DS のいずれかである場合のみです。

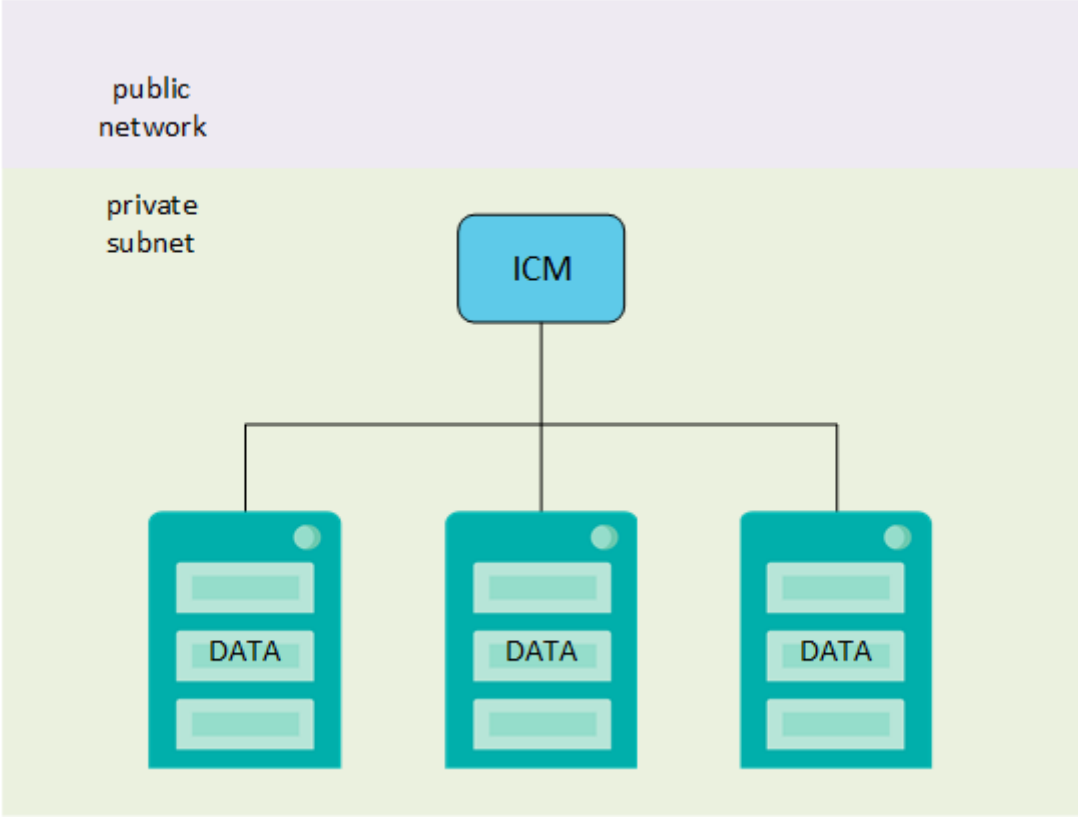
しかし、パブリック・インターネットから構成にまったくアクセスできないようにしたい場合もあります。その場合は、ICM を使用して、直接的なパブリック・アクセスを提供しないように、構成をプライベート・ネットワークに導入することができます。ICM 自体がそのネットワークに導入されている場合は通常の方法でプロビジョニングと導入を行うことができますが、そうでない場合は、ICM にそのネットワークへのアクセスを提供するノードをパブリック・ネットワークの外部でプロビジョニングする必要があります。このノードを要塞ホストと呼びます。これらの要素を踏まえて、プライベート・ネットワークを使用するには以下の 3 つの手法があります。

- 既存のプライベート・ネットワーク内で ICM をインストールして実行します。いくつかのフィールドを使用して (プロバイダによって異なるフィールドもあります)、これを ICM に指示します。
- プライベート・ネットワークへのアクセスを ICM に提供する要塞ホストを ICM でプロビジョニングし、以下のいずれかに構成をプロビジョニングおよび導入します。
  - ICM によって作成されるプライベート・ネットワーク。
  - 既存のプライベート・ネットワーク (適切なフィールドを使用して記述)。

4.11.1 既存のプライベート・ネットワーク内での導入

以下の図のように、ICM を既存のプライベート・ネットワークに導入し、そのネットワークでプロビジョニングと導入を行う場合は、導入する構成の既定値ファイルと定義ファイルにフィールドを追加する必要があります。

図 4-3: プライベート・サブネット内に導入された ICM



既存のプライベート・ネットワークで導入を行うには、以下の手順に従います。

1. プライベート・ネットワーク内にあるノードにアクセスできるようにします。これには、VPN または中間ホストの使用が必要な場合があります。
2. “ICM の使用” の章にある ″ICM の起動″ の説明に従って、そのノードに Docker と ICM をインストールします。
3. 以下のフィールドを defaults.json ファイルに追加します。

```
"PrivateSubnet": "true",
"net_vpc_cidr": "10.0.0.0/16",
"net_subnet_cidr": "10.0.2.0/24"
```

net_vpc_cidr フィールドと net_subnet_cidr フィールド (サンプル値が示されている) では、プライベート・ネットワークの CIDR とそのネットワーク内のノードのサブネットの CIDR を指定します。

4. 適切な共通フィールドとプロバイダ固有のフィールドを以下のように defaults.json ファイルに追加します。

プロバイダ	キー	説明
すべて	PrivateSubnet	true に設定する必要があります。
	net_vpc_cidr	プライベート・ネットワークの CIDR
	net_subnet_cidr	プライベート・ネットワーク内の ICM ノードのサブネットの CIDR (″メモ″ を参照)
GCP	Network	Google の VPC
	Subnet	Google のサブネットワーク
Azure	ResourceGroupName	AzureRM のリソース・グループ
	VirtualNetworkName	AzureRM の仮想ネットワーク
	SubnetName	AzureRM のサブネット (″メモ″ を参照)
AWS (″メモ″ を参照)	VPCId	AWS の VPC ID
	SubnetIds	AWS のサブネット ID のコンマ区切りリスト。Zone フィールドによって指定された要素ごとに 1 つずつサブネット ID を追加します。
Tencent	VPCId	Tencent の VPC ID
	SubnetIds	Tencent のサブネット ID のコンマ区切りリスト。Zone フィールドによって指定された要素ごとに 1 つずつサブネット ID を追加します。

注釈 ICM は Azure 上で、SubnetName で指定されたサブネットにセキュリティ・グループを割り当てますが、これがサブネット上の無関係のマシンの動作に影響を与える可能性があります。このため、定義ファイル内でそれぞれのエントリについて、専用のサブネット (一意の SubnetName および対応する net_subnet_cidr で指定) を指定する必要があります (ただし、ResourceGroupName と VirtualNetworkName は既定値ファイル内で指定します)。以下のセクションの説明に従って要塞ホストを導入する場合、これには BH の定義も含まれます。

AWS 上の既存のプライベート VPC 内に InterSystems IRIS を導入するには、ICM を導入して使用できるノードをその VPC 内に作成する必要があります。VPC の外部からこの ICM ホストにアクセスする場合は、ICM が独自のものを作成する代わりに使用するルート・テーブルとインターネット・ゲートウェイを指定できます。そのためには、RouteTableId フィールドと InternetGatewayId フィールドを defaults.json ファイルに追加します。以下に例を示します。

```
"RouteTableID": "rtb-00bef388a03747469",
"InternetGatewayId": "igw-027ad2d2b769344a3"
```

GCP でプロビジョニングを行う場合、net_subnet_cidr フィールドは、禁令的ではなく叙述的です。これは、ノードのサブネットを含むアドレス空間である必要があると同時に、ネットワーク内の他のすべてのノードが、導入された構成にアクセスできる必要があります。

5. icm provision および icm run を使用して、構成のプロビジョニングと導入を行います。

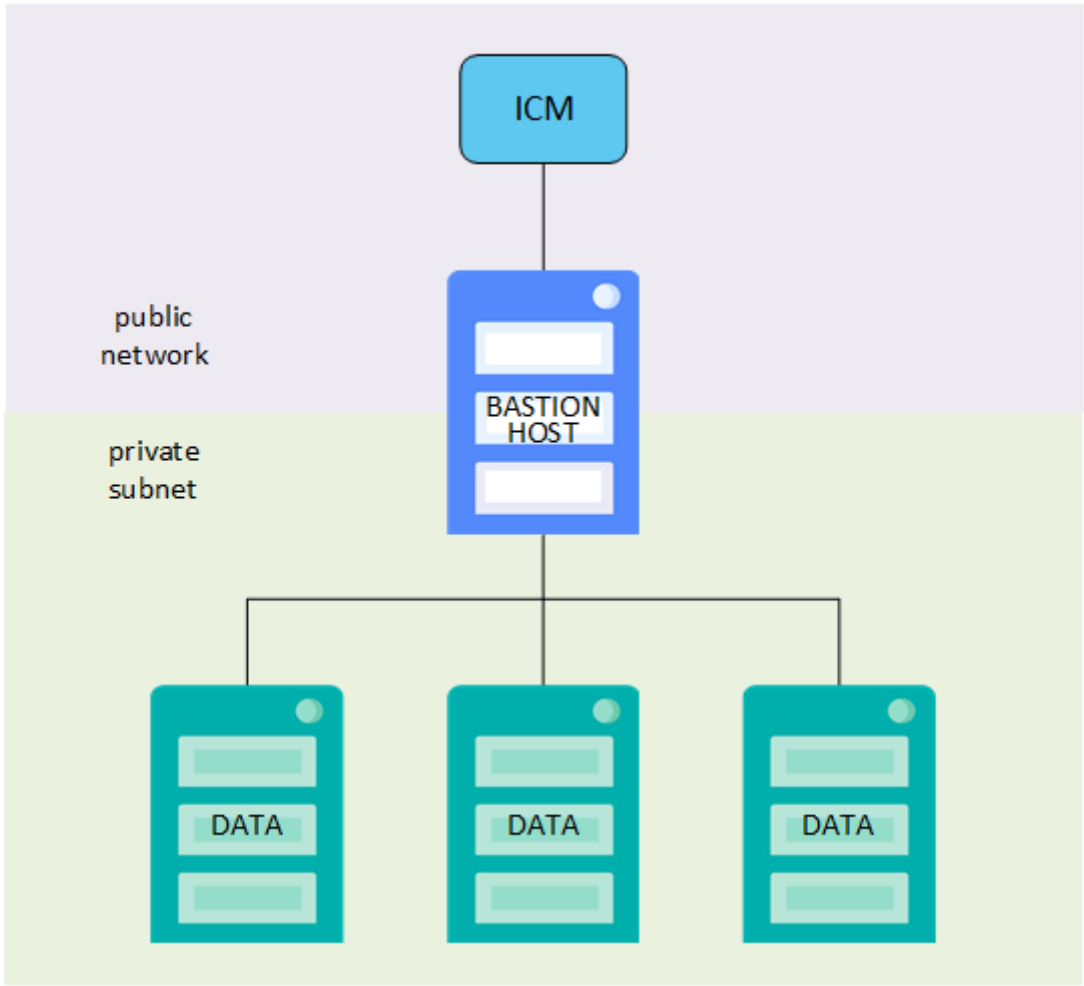
プライベート・ネットワークで導入を行う際には、以下の点に留意してください。

- 管理ポータルなどの Web ページをプライベート・ネットワーク内のノードで表示するには、同様にプライベート・ネットワーク内にあるブラウザか、プロキシまたは VPN が構成されているブラウザが必要です。
- ICM コマンドの出力に表示される DNS 名は、単にローカル IP アドレスのコピーです。
- 現在、プライベート・ネットワークを複数のリージョンまたはプロバイダにわたって導入することはサポートされていません。

4.11.2 要塞ホストを使用したプライベート・ネットワークへの導入

既定値ファイルで PrivateSubnet フィールドを true に設定しているにもかかわらず、既存のネットワークを使用するために必要なフィールドを含めていない場合、ICM によってプライベート・ネットワークが作成されます。ただし、ICM は、割り振ったマシンを構成したり、それらのマシンとやり取りしたりすることができないので、この状態ではプロビジョニング・フェーズを完了することはできません。作成したプライベート・ネットワーク上のノードとやり取りできるようにするために、ICM ではオプションで要塞ホストを作成できます。これは、プライベート・サブネットとパブリック・ネットワークの両方に属し、それらの間の通信を仲介できるホスト・ノードです。

図 4-4: 要塞ホストを使用してプライベート・ネットワークの外部に導入された ICM



プライベート・ネットワークと、そのネットワークへのアクセスを ICM に提供する要塞ホストを作成するには、タイプ BH の 1 つのノードの定義を definitions.json ファイルに追加します。以下に例を示します。

```
{
    "Role": "DATA",
    "Count": "3"
},
{
    "Role": "BH",
    "Count": "1",
    "StartCount: 4"
}
```

要塞ホストを導入し、既存のプライベート・ネットワークで使用するには、前述のように BH 定義を定義ファイルに追加し、ネットワークを指定するために必要なフィールドを既定値ファイルに含めます (前のセクションを参照)。BH ノードの定義が definitions.json に含まれている場合、ICM は自動的に "PrivateSubnet" オプションを "true" に設定します。

要塞ホストには SSH を使用してアクセスでき、ユーザは要塞ホストを介して SSH コマンドをプライベート・ネットワークにトンネリングすることができます。この手法を使用することで、ICM はプライベート・ネットワーク内の計算インスタンスを外部から割り振って構成し、プロビジョニングを正常に完了することができます。以下に例を示します。

```
$ icm inventory
Machine             IP Address     DNS Name                      Region   Zone
-------             ----------     --------                      ------   ----
Acme-BH-TEST-0004   35.237.125.218 218.125.237.35.bc.google.com us-east1 b
Acme-DATA-TEST-0001 10.0.0.2       10.0.0.2                      us-east1 b
Acme-DATA-TEST-0002 10.0.0.3       10.0.0.3                      us-east1 b
Acme-DATA-TEST-0003 10.0.0.4       10.0.0.4                      us-east1 b
```

構成が導入されたら、ノードに対して ssh コマンドを実行できます。以下に例を示します。

```
# icm ssh -role DATA -interactive
ubuntu@ip-10.0.0.2:~$
```

ただし、コマンドが実行されていることを確認すると、要塞ホストを介してルーティングされていることがわかります。

```
$ icm ssh -role DATA -interactive -verbose
ssh -A -t -i /Samples/ssh/insecure -p 3022 ubuntu@35.237.125.218
ubuntu@ip-10.0.0.2:~$
```

一方、他のコマンドが成功するためには、ICM は SSH 以外のポートおよびプロトコルへのアクセスを必要とします。そのために、ICM は、Docker、JDBC、および HTTP について要塞ホストとクラスタ内のノードとの間のトンネルを構成します。これにより、icm run、icm exec、icm sql などのコマンドを正常に実行することができます。

要塞ホストを導入する際には、以下の点に留意してください。

- 構成の管理ポータルのアドレスは要塞ホストのものです。
- セキュリティ上の理由から、秘密鍵が要塞ホストに保存されることはありません。
- ICM コマンドの出力に表示される DNS 名は、単にローカル IP アドレスのコピーです。
- 要塞ホストを含む導入環境でのロード・バランサのプロビジョニングはサポートされていません。
- 現在、マルチリージョン導入環境 ("複数のリージョンまたはプロバイダにわたる導入" を参照) および分散管理モード (付録 "ICM 導入環境の共有" を参照) での要塞ホストの使用はサポートされていません。

  注釈　Google ポータルでカスタム VPC を作成する際には、既定のサブネットを作成する必要があります。要塞ホストを使用してプロビジョニングを行い、ICM によって作成されたサブネットを使用する場合は、プロビジョニングの前にこの既定のサブネットを削除する必要があります (または、既定のアドレス空間 10.0.0.0/16 と競合しないアドレス空間を提供します)。

4.12 InterSystems API Manager の導入

InterSystems API Manager (IAM) では、一元化されたゲートウェイを介してトラフィックをルーティングし、API 要求を適切なターゲット・ノードに転送することで、Web ベースの API との間のトラフィックの監視と制御を行うことができます。IAM の詳細は、"IAM ガイド" を参照してください。

IAM が ICM 導入環境に組み込まれるのは、定義ファイルで CN ノードを定義し、値 **true** で **IAM** フィールドを追加し、**IAMImage** フィールドを使用して InterSystems IRIS **iam** イメージを指定した場合です。以下に例を示します。

```
[
    {
        "Role": "DATA",
        "Count": "1",
        "LicenseKey": "ubuntu-sharding-iris-with-iam.key"
    },
    {
        "Role": "CN",
        "Count": "1",
        "IAM": "true",
        "IAMImage": "intersystems/iam:2.0"
    }
]
```

IAM コンテナは、導入フェーズで導入されます ("icm run コマンド" を参照)。**PostgresImage** フィールドで Postgres イメージを指定して、オプションで Postgres コンテナを導入することもできます。既定値は "一般パラメータ" に示されています。

導入に成功すると、下の例のようなメッセージが表示されます。

```
$ icm run
...
-> IAM Portal available at: http://112.97.196.104.google.com:8080/overview#
```

IAM は、IAM 対応ライセンスを取得するため、最初の (または唯一の) **iris** コンテナ内の InterSystems IRIS インスタンス (例えば、シャード・クラスタ内のノード 1) にアタッチします。ミラーリングが有効な場合、これが最初の (または唯一の) フェイルオーバー・ペアのプライマリになります。

注釈　IAM はコンテナレス・モードでは導入できません。

4.13 ICM でのモニタリング

ICM 導入環境で InterSystems IRIS インスタンスを監視するために、System Alerting and Monitoring クラスタ・モニタリング・ソリューションを導入できます。

サードパーティのモニタリング・パッケージを ICM 構成の一部として導入することもできます。

4.13.1 System Alerting and Monitoring

System Alerting and Monitoring (SAM) は、InterSystems IRIS® データ・プラットフォーム用のクラスタ・モニタリング・ソリューションです。InterSystems IRIS ベースのアプリケーションが動作する構成やプラットフォームに関係なく、SAM を使用して監視することができます。SAM の詳細は、"System Alerting and Monitoring Guide" を参照してください。

SAM が ICM 導入環境に組み込まれるのは、定義ファイルに SAM ノードを含めた場合です。DockerImage フィールドは必須で、InterSystems IRIS **sam** イメージを指定する必要があります。以下に例を示します。

```
[
    {
        "Role": "DM",
        "Count": "4",
        "LicenseKey": "ubuntu-sharding-iris.key"
    },
    {
        "Role": "SAM",
        "Count": "1",
        "DockerImage": "intersystems/sam:2.0"
    }
]
```

SAM アプリケーションは、5 つのコンテナで構成されます。SAM マネージャ・コンテナは、導入フェーズで導入されます (″icm run コマンド″ を参照)。

他の 4 つのコンテナ (Prometheus、Alertmanager、Grafana、Nginx) は、プロビジョニング・フェーズで導入されます (″icm provision コマンド″ を参照)。これらのコンテナの導入元のイメージは、PrometheusImage、AlertmanagerImage、GrafanaImage、NginxImage フィールドを使用して指定できます。これらの既定値は、″一般パラメータ″ に示されています。

導入に成功すると、下の例のようなメッセージが表示されます。

```
$ icm run
...
-> SAM Portal available at: http://112.97.196.104.google.com:8080/api/sam/app/index.csp#
```

注釈　SAM はコンテナレス・モードでは導入できません。

4.13.2 ICM でのサードパーティ・モニタリングの導入

サードパーティの好みのモニタリング・パッケージ (または、その他のサードパーティ・パッケージ) を ICM 構成の一部として導入できます。以下の例は、icm ssh コマンドを使用して、導入環境内のすべてのホストに Weave Scope モニタリングを追加する方法を示しています。

```
icm ssh -command "sudo curl -L git.io/scope -o /usr/local/bin/scope 2>&1"
icm ssh -command "sudo chmod +x /usr/local/bin/scope"
icm ssh -command "sudo /usr/local/bin/scope launch 2>&1"
```

これらのコマンドの実行後、icm inventory コマンドで表示された任意のホスト上のポート 4040 (**http://hostname:4040**) を経由して、Weave Scope にアクセスできます。

重要　この簡単な例は、説明のみを目的としています。Weave Scope は認証が不要であるため、本質的に安全ではありません。ファイアウォールでポート 4040 が開かれている場合、その URL を知るすべての人がコンテナにアクセスできます。十分に安全であると確信できる場合を除き、プライベート・ネットワークの外部にサードパーティ・パッケージを導入しないでください。

4.14 ICM のトラブルシューティング

ICM 操作中にエラーが発生した場合、エラーに関する情報を確認できるログ・ファイルを示すメッセージを ICM は表示します。ICM の導入を開始する前に、″ログ・ファイルとその他の ICM ファイル″ で説明されている、ログ・ファイルとその場所をよく理解しておいてください。

InterSystems IRIS イメージのコンテナ・イメージを作成して実行する際の重要な考慮事項については、以降のトピックに加えて、"コンテナ内での InterSystems IRIS の実行" の "Docker/InterSystems IRIS に関するその他の考慮事項" を参照してください。

- ホスト・ノードの再起動とリカバリ
- 時刻スキューの訂正
- ICM でのタイムアウト
- Docker ブリッジ・ネットワーク IP アドレス範囲の競合
- Weave ネットワーク IP アドレス範囲の競合
- 巨大なページ

4.14.1 ホスト・ノードの再起動とリカバリ

計画外停止、またはクラウド・プロバイダ (予防メンテナンスの場合など) やユーザ (コスト削減のためなど) による計画的な措置によってクラウド・ホスト・ノードがシャットダウンされ、再起動された場合、その IP アドレスとドメイン名が変更されることがあります。これにより、ICM と導入されたアプリケーション (InterSystems IRIS を含む) の両方で問題が生じます。

この動作は、クラウド・プロバイダによって異なります。GCP および Azure では、既定で、ホスト・ノードの再起動後も IP アドレスとドメイン名が保持されますが、AWS と Tencent ではこの機能はオプションです ("Elastic IP 機能" を参照)。

ホスト・ノードがシャットダウンされる理由としては、以下のものがあります。

- 計画外停止
  - 停電
  - カーネル・パニック
- プロバイダが開始した予防メンテナンス
- ユーザが開始したコスト削減戦略

ホスト・ノードを意図的にシャットダウンする方法としては、以下のものがあります。

- クラウド・プロバイダのユーザ・インタフェースの使用
- ICM の使用 :

```
icm ssh -command 'sudo shutdown'
```

4.14.1.1 Elastic IP 機能

AWS の Elastic IP 機能では、ホスト・ノードの再起動後も IP アドレスとドメイン名が保持されます。ICM では、この機能は既定で無効になっています。その理由の 1 つは、停止しているマシンに追加の料金がかかるためです (実行中のマシンにはかかりません)。この機能を有効にするには、defaults.json ファイルで ElasticIP フィールドを true に設定します。必ず、お使いのプロバイダの機能を確認してください (AWS ドキュメントの "Elastic IP アドレス" または Tencent ドキュメントの "Elastic Public IP" を参照してください)。

4.14.1.2 リカバリと再起動の手順

ホスト・ノードの IP アドレスとドメイン名が変更された場合、ICM はノードと通信できなくなります。そのため、手動更新に続けて、クラスタに対する更新が必要になります。ICM によって導入される Weave ネットワークには、分散型の検出サービスが含まれています。つまり、少なくとも 1 つのホスト・ノードが元の IP アドレスを保持していれば、他のホスト・ノードがそれに到達し、相互とのすべての接続を再確立することができます。ただし、クラスタ内のすべてのホスト・ノードの IP ア

ドレスが変更された場合、Weave ネットワーク内のすべてのノードを有効な IP アドレスに接続するには、追加の手順が必要です。

手動更新の手順は以下のとおりです。

1. クラウド・プロバイダの Web コンソールに移動し、そこでインスタンスを探します。それぞれの IP アドレスとドメイン名を記録します。以下に例を示します。

ノード	IP アドレス	ドメイン名
ANDY-DATA-TEST-0001	54.191.233.2	ec2-54-191-233-2.amazonaws.com
ANDY-DATA-TEST-0002	54.202.223.57	ec2-54-202-223-57.amazonaws.com
ANDY-DATA-TEST-0003	54.202.223.58	ec2-54-202-223-58.amazonaws.com

2. instances.json ファイルを編集し (“基本的な ICM の要素” の章の ″インスタンス・ファイル″ を参照)、各インスタンスの IPAddress フィールドと DNSName フィールドを更新します。以下に例を示します。

```
"Label" : "SHARDING",
"Role" : "DATA",
"Tag" : "TEST",
"MachineName" : "ANDY-DATA-TEST-0001",
"IPAddress" : "54.191.233.2",
"DNSName" : "ec2-54-191-233-2.amazonaws.com",
```

3. icm inventory コマンドを使用して、値が正しいことを確認します。

```
$ icm inventory
Machine                 IP Address    DNS Name                        Region   Zone
-------                 ----------    --------                        ------   ----
ANDY-DATA-TEST-0001 54.191.233.2  ec2-54-191-233-2.amazonaws.com  us-east1 b
ANDY-DATA-TEST-0002 54.202.223.57 ec2-54-202-223-57.amazonaws.com us-east1 b
ANDY-DATA-TEST-0003 54.202.223.58 ec2-54-202-223-58.amazonaws.com us-east1 b
```

4. icm ps コマンドを使用して、ホスト・ノードに到達できることを確認します。

```
$ icm ps -container weave
Machine                   IP Address      Container   Status   Health    Image
-------                   ----------      ---------   ------   ------    -----
ANDY-DATA-TEST-0001   54.191.233.2    weave       Up                 weaveworks/weave:2.0.4
ANDY-DATA-TEST-0002   54.202.223.57   weave       Up                 weaveworks/weave:2.0.4
ANDY-DATA-TEST-0003   54.202.223.58   weave       Up                 weaveworks/weave:2.0.4
```

5. すべての IP アドレスが変更されている場合は、この例の 54.191.233.2 など、新しいアドレスのいずれかを選択します。その後、以下のように、icm ssh コマンドを使用して各ノードをこの IP アドレスに接続します。

```
$ icm ssh -command "weave connect --replace 54.191.233.2"
Executing command 'weave connect 54.191.233.2' on host ANDY-DATA-TEST-0001...
Executing command 'weave connect 54.191.233.2' on host ANDY-DATA-TEST-0002...
Executing command 'weave connect 54.191.233.2' on host ANDY-DATA-TEST-0003...
...executed on ANDY-DATA-TEST-0001
...executed on ANDY-DATA-TEST-0002
...executed on ANDY-DATA-TEST-0003
```

4.14.2 時刻スキューの訂正

ICM コンテナ内のシステム時刻が標準時刻と数分以上異なる場合は、各種クラウド・プロバイダが ICM からの要求を拒否する可能性があります。これは、コンテナが起動時（初期、または停止や一時停止の後）に NTP サーバに到達できない場合に起こることがあります。このエラーは、terraform.err ファイルに以下のような差異として示されます。

```
Error refreshing state: 1 error(s) occurred:

    # icm provision
    Error: Thread exited with value 1
    Signature expired: 20170504T170025Z is now earlier than 20170504T171441Z (20170504T172941Z   15
min.)
    status code: 403, request id: 41f1c4c3-30ef-11e7-afcb-3d4015da6526 doesn't run for a period of time
```

解決法は、NTP を手動で実行することです。例を以下に示します。

```
ntpd -nqp pool.ntp.org
```

その後、時刻が正しくなったことを確認します。（"ICM の起動" の --cap-add オプションに関する説明も参照してください。）

4.14.3 ICM でのタイムアウト

ターゲット・システムに過大な負荷がかかっている場合、ICM のさまざまな操作がタイムアウトになることがあります。これらのタイムアウトの多くは、ICM の直接の制御下にありません（例えば、クラウド・プロバイダから制御される）。SSH 接続や JDBC 接続など、その他の操作は何度か再試行されます。

SSH のタイムアウトは、場合によっては正しく認識されないことがあります。例えば、以下の例では、SSH タイムアウトは基礎となるライブラリからの一般例外として現れています。

```
# icm cp -localPath foo.txt -remotePath /tmp/
2017-03-28 18:40:19 ERROR Docker:324 - Error:
java.io.IOException: com.jcraft.jsch.JSchException: channel is not opened.
2017-03-28 18:40:19 ERROR Docker:24 - java.lang.Exception: Errors occurred during execution; aborting
 operation
        at com.intersystems.tbd.provision.SSH.sshCommand(SSH.java:419)
        at com.intersystems.tbd.provision.Provision.execute(Provision.java:173)
        at com.intersystems.tbd.provision.Main.main(Main.java:22)
```

この場合に推奨される対処法は、操作を再試行することです（最も近い原因を特定して解決した後）。

セキュリティ上の理由から、ICM はアイドル・セッションの既定の SSH タイムアウトを 10 分に設定します（60 秒 x 10 回の再試行）。これらの値は、/etc/ssh/sshd_config ファイル内で以下のフィールドを修正することによって変更できます。

```
ClientAliveInterval 60
ClientAliveCountMax 10
```

4.14.4 Docker ブリッジ・ネットワーク IP アドレス範囲の競合

コンテナ・ネットワークの場合、Docker では、既定で、サブネット 172.17.0.0/16 でブリッジ・ネットワークが使用されます（Docker ドキュメントの "Use bridge networks" を参照）。このサブネットを既にネットワークで使用している場合は、競合が発生して、Docker が起動しなくなったり、導入したホスト・ノードにアクセスできなくなったりすることがあります。この問題は、ICM コンテナをホストしているマシンと InterSystems IRIS クラスタ・ノードのどちらかまたは両方で生じる可能性があります。

これを解決するには、使用している IP アドレスと競合しない範囲にサブネットを割り当て直すように、Docker 構成ファイルのブリッジ・ネットワークの IP 構成を編集できます (ユーザの組織の IT 部門がこの値の指定を支援できる場合があります)。この変更を行うには、Docker デーモン構成ファイルに以下のような行を追加します。

```
"bip": "192.168.0.1/24"
```

ICM コンテナでこの問題が生じる場合は、コンテナのホストで /etc/docker/daemon.json ファイルを編集します。導入された構成内のホスト・ノードでこの問題が生じる場合は、ICM コンテナ内の /ICM/etc/toHost/daemon.json ファイルを編集します。既定でこのファイルには前述の例で示した値が含まれています。この値によって、PreExisting 以外の導入タイプで問題を回避できる可能性があります。

daemon.json ファイルのコンテンツに関する詳細は、Docker ドキュメントの "Daemon configuration file" に記載されています。また、"Configure and troubleshoot the Docker daemon" も参照してください。

4.14.5 Weave ネットワーク IP アドレス範囲の競合

既定では、Weave ネットワークは IP アドレス範囲 10.32.0.0/12 を使用します。これが既存のネットワークと競合する場合は、ログ・ファイル installWeave.log に以下のようなエラーが表示されることがあります。

```
Network 10.32.0.0/12 overlaps with existing route 10.0.0.0/8 on host
ERROR: Default --ipalloc-range 10.32.0.0/12 overlaps with existing route on host.
You must pick another range and set it on all hosts.
```

このエラーは、指定されたマシンが他のソフトウェアやローカル・ポリシーをサポートするためにカスタム・ネットワーク構成を行っている場合に、プロバイダ PreExisting で発生することが考えられます。他のネットワークを無効にしたり移動したりすることができない場合は、以下の手順を使用して Weave 構成を代わりに変更できます。

1. ICM コンテナでローカルにある以下のファイルを編集します。

   ```
   /ICM/etc/toHost/installWeave.sh
   ```

2. 文字列 weave launch を含む行を見つけます。Weave と既存のネットワークの間に重複が生じる危険がないことが確実ならば、以下に示す下線の付いたテキストを追加することによって、既定の範囲を引き続き使用するように Weave に強制できます。

   ```
   sudo /usr/local/bin/weave launch --ipalloc-range 10.32.0.0/12 --password $2
   ```

   以下のように、Weave を別のプライベート・ネットワークに単に移動することもできます。

   ```
   sudo /usr/local/bin/weave launch --ipalloc-range 172.30.0.0/16 --password $2
   ```

3. ファイルを保存します。
4. クラスタの再プロビジョニングを行います。

4.14.6 巨大なページ

一部のアーキテクチャでは、InterSystems IRIS メッセージ・ログに以下のようなエラーが表示される場合があります。

```
0 Automatically configuring buffers
1 Insufficient privileges to allocate Huge Pages; nonroot instance requires CAP_IPC_LOCK capability
for Huge Pages.
2 Failed to allocate 1316MB shared memory using Huge Pages. Startup will retry with standard pages. If
 huge pages
  are needed for performance, check the OS settings and consider marking them as required with the
InterSystems IRIS
  'memlock' configuration parameter.
```

icm run コマンドに以下のオプションを指定することによって、このエラーを解消できます。

```
-options "--cap-add IPC_LOCK"
```

A
コンテナレスの導入

重要　　InterSystems IRIS リリース 2023.3 以降、InterSystems Cloud Manager (ICM) は非推奨になっています。今後のバージョンからは削除される予定です。

ICM を使用して、クラウド・インフラストラクチャをプロビジョニングし、そのインフラストラクチャにコンテナ化されていない InterSystems IRIS インスタンスをインストールする場合、または PreExisting クラスタに InterSystems IRIS をインストールする場合、コンテナレス・モードでこれを行うことができます。

基本的に、コンテナレス・モードでは、ICM による InterSystems IRIS のコンテナ化された導入が、従来のキットからの直接のインストールに置き換えられる一方で、ICM のプロビジョニングおよび導入プロセスのその他すべての手順は維持されます。これは、ICM にコマンドを追加し、他のいくつかのコマンドを適応させることで実現できます。コンテナレス・モードでは、プロビジョニングされたノードに Docker がインストールされることはなく、またそういったノードにコンテナを導入するために icm run コマンドを使用することはできません。

ICM では、コンテナレス・モードで InterSystems IRIS の非 root インストールを実行できます。詳細は、"コンテナレス・モードでの非 root インストール" を参照してください。

A.1 コンテナレス導入プラットフォーム

コンテナレスの導入でサポートされているオペレーティング・システムには、次のようなものがあります。

- Ubuntu 20.04 以降
- Red Hat Enterprise Linux 8.3 以降のバージョン 8 リリース、9.0 以降のバージョン 9 リリース

A.2 コンテナレス・モードの有効化

コンテナレス・モードを有効にするには、defaults.json ファイルに Containerless フィールドを追加し、値を true に設定します。以下に例を示します。

```
{
    "Containerless": "true",
    "Provider": "AWS",
    "Label": "Sample",
    "Tag": "TEST"
    "LicenseDir": "/Samples/license/",
    "Credentials": "/Samples/AWS/sample.credentials",
    ...
}
```

A.3 InterSystems IRIS のインストール

プロビジョニングされたノードに、選択したインストール・キットを使用して InterSystems IRIS をインストールするには、icm install コマンドを使用します。このコマンドは、コンテナ・モードには存在しません。キットは、KitURL フィールドによって指定します。このフィールドにインストール・キットへのパスを指定し、defaults.json か definitions.json のいずれかに追加できます。指定するキットは、以下のすべてを満たしている必要があります。

- InterSystems IRIS のインストール先のノードからアクセスできる (ICM コンテナ自体からは必ずしもアクセス可能である必要はありません)
- 64 ビットの Linux キットである
- gzip 圧縮の tar ファイルである

例えば、定義ファイルは以下のようになります。

```
[
    {
        "Role": "DM",
        "Count": "1",
        "DataVolumeSize": "50",
        "InstanceType": "m4.xlarge",
        "KitURL": "http://kits.acme.com/iris/2022.2.0/unix/IRIS-2022.2.0.792.0-lnxrhx64.tar.gz"
    },
    {
        "Role": "AM",
        "Count": "2",
        "StartCount": "2",
        "LoadBalancer": "true",
        "KitURL": "http://kits.acme.com/iris/2022.2.0/unix/IRIS-2022.2.0.792.0-lnxrhx64.tar.gz"
    }
]
```

既定値ファイルは以下のようになります。

```
{
    "Containerless": "true",
    "KitURL": "http://kits.acme.com/iris/2022.2.0/unix/IRIS-2022.2.0.792.0-lnxrhx64.tar.gz"
    "Provider": "AWS",
    "Label": "Sample",
    "Tag": "TEST"
    "LicenseDir": "/Samples/license/",
    "Credentials": "/Samples/AWS/sample.credentials",
    ...
}
```

注釈　KitURL を、プロビジョニングされたノードにコピーするローカル・ファイルへの参照にすることができます。これは、状況によっては便利な場合があります。例えば、以下の KitURL を既定値ファイルに追加できます。

```
"KitURL": "file://tmp/IRIS-2022.2.0.792.0-lnxrhx64.tar.gz"
```

そのうえで、icm install コマンドを実行する前に、プロビジョニングされたノードに、icm scp コマンドを使用してこのキットをコピーします。例えば、以下のようになります。

```
icm scp -localFile IRIS-2022.2.0.792.0-lnxrhx64.tar.gz -remoteFile /tmp
```

icm install コマンドの実行時に、ICM によって InterSystems IRIS が、指定されたキットから各該当ノードにインストールされ、以下のような出力となります。

```
Downloading kit on Acme-DM-TEST-0001...
Downloading kit on Acme-AM-TEST-0002...
Downloading kit on Acme-AM-TEST-0003...
...downloaded kit on Acme-AM-TEST-0002
...downloaded kit on Acme-AM-TEST-0003
...downloaded kit on Acme-DM-TEST-0001
Installing kit on Acme-AM-TEST-0003...
Installing kit on Acme-DM-TEST-0001...
Installing kit on Acme-AM-TEST-0002...
...installed kit on Acme-AM-TEST-0002
...installed kit on Acme-DM-TEST-0001
...installed kit on Acme-AM-TEST-0003
Starting InterSystems IRIS on Acme-DM-TEST-0001...
Starting InterSystems IRIS on Acme-AM-TEST-0002...
Starting InterSystems IRIS on Acme-AM-TEST-0003...
...started InterSystems IRIS on Acme-AM-TEST-0002
...started InterSystems IRIS on Acme-AM-TEST-0003
...started InterSystems IRIS on Acme-DM-TEST-0001
Management Portal available at: http://172.16.110.14:52773/csp/sys/UtilHome.csp
```

注釈　UserCPF フィールドと構成マージ・ファイルを使用して、同じキットから複数のインスタンスを異なる構成設定でインストールすることができます。詳細は、“ICM リファレンス” の章にある ″カスタマイズされた InterSystems IRIS 構成を使用した導入″ を参照してください。

A.4 InterSystems IRIS の再インストール

コンテナレスの導入プロセスの柔軟性と回復力を可能な限り高めるために、icm install コマンドは完全に再入可能であり、同じ導入環境について複数回発行できます。この点で、“ICM の使用” の章にある ″サービスの再導入″ に記載されている icm run コマンドと似ています。

icm install コマンドを繰り返すと、ICM は、インストールされているインスタンスを停止してアンインストールし、その後、指定されたキットから InterSystems IRIS を再度インストールします。このコマンドを繰り返す理由として以下のいずれかが考えられます。

- 既存のインスタンスを再インストールする。

  インストールされている InterSystems IRIS インスタンスを同じキットから新しいものに置き換えるには、最初にコンテナを導入した元の icm run コマンドを繰り返します。LicenseDir フィールドで指定されたディレクトリ内のライセンスを更新した場合など、再インストールを必要とする変更を定義ファイル内で行った場合に、この作業を行うことがあります。

- ″インフラストラクチャの再プロビジョニング″ の説明に従ってインフラストラクチャに追加したノードに InterSystems IRIS をインストールする。

  インフラストラクチャにノードを追加した後に icm install コマンドを繰り返すと、既存のノード上のインスタンスは前述のように再インストールされ、新しいノードには新しいインスタンスがインストールされます。これにより、必要に応じて、既存のノードを新しい導入トポロジに合わせて再構成することができます。

- 導入エラーを解決する。

  ネットワークの遅延や中断など、ICM の制御下にない要因によって 1 つ以上のノードで icm install コマンドが失敗した場合、コマンドを再度発行できます。ほとんどの場合、繰り返し試行することで導入が正常に完了します。どうしてもエラーが解決されない場合は、コマンドを再度発行する前に、手動による操作が必要になることがあります (いずれかの構成ファイル内のエラーが原因である場合など)。

A.5 InterSystems IRIS のアンインストール

コンテナ・モードには存在しない icm uninstall コマンドは、コンテナレス・モードでは、導入環境内のすべての InterSystems IRIS インスタンスの停止およびアンインストールに使用できます (オプションを指定しない場合)。通常どおり、-role オプションと -machine オプションを使用すると、特定のロールまたはノードにコマンドを限定できます。以下はその例です。

```
icm uninstall
```

これは、導入環境内のすべてのノード上の InterSystems IRIS をアンインストールします。一方、次の例もご覧ください。

```
icm uninstall -role AM
```

これは、AM ノード上の InterSystems IRIS のみをアンインストールします。

A.6 コンテナレス・モードのその他のコマンド

以下に示したように、いくつかのコンテナ・モード・コマンドは、コンテナレス・モードでも同じように、あるいは似たように動作します。これには、-machine オプションと -role オプションの使用も含まれます。

- icm ssh、icm scp、icm session、 icm sql

  これらのコマンドの動作は、コンテナ・モードとコンテナレス・モードでまったく同じです。

- icm ps

  以下の例は、コンテナレス・モードでの icm ps からの出力に含まれる列を示しています。

```
# icm ps -json
Machine            IP Address     Instance   Kit             Status   Health
-------            ----------     --------   ---             ------   ------
Acme-DM-TEST-0001  54.67.2.117    IRIS       2022.2.0.792.0  running  ok
Acme-AM-TEST-0002  54.153.96.236  IRIS       2022.2.0.792.0  running  ok
Acme-AM-TEST-0003  54.103.9.388   IRIS       2022.2.0.792.0  running  ok
```

  Instance フィールドには各インスタンスの名前が表示され (インターシステムズ提供のコンテナ内では常に **IRIS** です)、Kit フィールドにはインストール元のキットが表示されます。Status の値としては、running、down、および sign-on inhibited があります。Health の値としては、ok、warn、および alert があります。

  注釈　コンテナレス・モードでの導入時に icm install コマンドの前に icm ps コマンドを使用した場合、Status フィールドには値 not installed が表示されます。

- icm stop、icm start

  icm stop コマンドおよび icm start コマンドは、すべての InterSystems IRIS インスタンスまたは指定されたインスタンスで、iris stop コマンドおよび iris start コマンドを実行します (″システム管理ガイド″ の “InterSystems IRIS 複数インスタンスの使用法” の章にある ″InterSystems IRIS インスタンスの制御″ を参照)。

- icm upgrade

  コンテナレス・モードでは、icm upgrade は以下を実行します。

  - KitURL フィールドで指定された InterSystems IRIS キットをダウンロードします。
  - iris stop を使用して、現在の InterSystems IRIS インスタンスを停止します。
  - 現在の InterSystems IRIS インスタンスをアンインストールします。

- KitURL で指定された InterSystems IRIS キットをインストールします。
- iris start を使用して、新しくインストールされた InterSystems IRIS インスタンスを起動します。

以下は、コンテナレス・モードでの icm upgrade コマンドからの出力を示しています。

```
# icm ps
Machine            IP Address     Instance   Kit             Status   Health
-------            ----------     --------   ---             ------   ------
Acme-DM-TEST-0001  54.67.2.117    IRIS       2022.2.0.792.0  running  ok
Acme-AM-TEST-0002  54.153.96.236  IRIS       2022.2.0.792.0  running  ok
Acme-AM-TEST-0003  54.103.9.388   IRIS       2022.2.0.792.0  running  ok

# icm upgrade
Downloading kit on Acme-DM-TEST-0001...
Downloading kit on Acme-AM-TEST-0002...
Downloading kit on Acme-AM-TEST-0003...
...downloaded kit on Acme-AM-TEST-0002
...downloaded kit on Acme-AM-TEST-0003
...downloaded kit on Acme-DM-TEST-0001
Stopping InterSystems IRIS on Acme-DM-TEST-0001...
Stopping InterSystems IRIS on Acme-AM-TEST-0003...
Stopping InterSystems IRIS on Acme-AM-TEST-0002...
...stopped InterSystems IRIS on Acme-DM-TEST-0001
...stopped InterSystems IRIS on Acme-AM-TEST-0002
...stopped InterSystems IRIS on Acme-AM-TEST-0003
Uninstalling InterSystems IRIS on Acme-AM-TEST-0003...
Uninstalling InterSystems IRIS on Acme-DM-TEST-0001...
Uninstalling InterSystems IRIS on Acme-AM-TEST-0002...
...uninstalled InterSystems IRIS on Acme-DM-TEST-0001
...uninstalled InterSystems IRIS on Acme-AM-TEST-0002
...uninstalled InterSystems IRIS on Acme-AM-TEST-0003
Installing kit on Acme-AM-TEST-0002...
Installing kit on Acme-DM-TEST-0001...
Installing kit on Acme-AM-TEST-0003...
...installed kit on Acme-AM-TEST-0002
...installed kit on Acme-DM-TEST-0001
...installed kit on Acme-AM-TEST-0003
Starting InterSystems IRIS on Acme-DM-TEST-0001...
Starting InterSystems IRIS on Acme-AM-TEST-0002...
Starting InterSystems IRIS on Acme-AM-TEST-0003...
...started InterSystems IRIS on Acme-AM-TEST-0002
...started InterSystems IRIS on Acme-AM-TEST-0003
...started InterSystems IRIS on Acme-DM-TEST-0001

# icm ps
Machine            IP Address     Instance   Kit             Status   Health
-------            ----------     --------   ---             ------   ------
Acme-DM-TEST-0001  54.67.2.117    IRIS       2022.2.1.417.0  running  ok
Acme-AM-TEST-0002  54.153.96.236  IRIS       2022.2.1.417.0  running  ok
Acme-AM-TEST-0003  54.103.9.388   IRIS       2022.2.1.417.0  running  ok
```

アップグレード後、既定値ファイルまたは定義ファイルで KitURL の値を更新して、アップグレードを反映させることができます。

A.7 コンテナレス・モードでの非 root インストール

サイトによっては、セキュリティ・ポリシーによりアプリケーションからシステムへの root アクセスが禁止されている場合があります。非 root の InterSystems IRIS インスタンスは、root アクセスを持たないユーザがインストールします。コンテナレス・モードを使用して非 root インスタンスをインストールする予定の場合は、操作を進める前に "インストール・ガイド" の "非 root ユーザとしてのインストール" で、root および非 root でのインストールとインスタンスの違いについて入念に確認してください。

A.7.1 必須の構成フィールド

非 root インストールの場合、defaults.json ファイルに以下のフィールドを含める必要があります (`containerless: true` に加えて)。

- nonroot フィールドを `true` に設定する必要があります。
- ISCInstalldir フィールドには、インストール・ユーザ (SSHUser フィールドで指定したユーザ) が書き込みアクセス権を持っているファイル・システムの場所 (既定のインストール・ディレクトリ /usr/irissys は該当しない) を指定する必要があります。例えば、SSHUser が **irisowner** の場合、以下のように指定できます。

  ```
  "ISCInstallDir": "/home/irisowner/irissys"
  ```

  プロバイダによっては、SSHUser フィールドに具体的なユーザ名を指定しなければならない場合があります。詳細は、“ICM リファレンス” の章の ″セキュリティ関連のパラメータ″ を参照してください。
- インストール・ユーザが /irissys (永続的な %SYS ボリュームの既定の場所) への書き込みアクセス権を持っていない場合は、ICM によってマウントされるストレージ・ボリュームのマウント・ポイントを上書きするために使用するフィールドを、そのユーザが書き込みアクセス権を持っている永続ストレージ上のマウント・ポイントに変更する必要があります。例えば、以下のように指定します。

  ```
  "DataMountPoint": "/home/irisowner/data",
  "WIJMountPoint": "/home/irisowner/wij",
  "Journal1MountPoint": "/home/irisowner/journal1",
  "Journal2MountPoint": "/home/irisowner/journal2"
  ```

  重要　非 root インストールの場合、ICM は外部ボリュームの割り当て、フォーマット、またはマウントを行いません。ユーザが責任を持って、マウント・ポイントのフィールドを使用して指定したディレクトリがアクセス可能な永続ストレージを表すようにする必要があります。

また、コンテナレス・モードの非 root 導入環境に WS ノードが含まれる場合、definitions.json ファイル内の WS 定義の KitURL フィールドで InterSystems IRIS キットを指定し、LicenseKey フィールドで InterSystems IRIS ライセンスを指定する必要があります。例えば、以下のように指定します。

```
{
  "Role": "WS",
  "Count": "1",
  "KitURL": "file://tmp/IRIS-2022.2.0.221.0-lnxubuntux64.tar.gz",
  "LicenseKey": "heterogenous-sharding-iris.key"
}
```

注釈　非 root ユーザは 1024 未満のポートにバインドできないため、コンテナレス非 root モードでは Web ゲートウェイ・ポートが 80 から 52773 に移動されています。例えば、WS ノード上の Web ゲートウェイの URL は、以下のようになります。

```
http://133.98.229.35.google.com:52773/csp/bin/Systems/Module.cxw
```

WebServerPort パラメータは、すべての導入環境の InterSystems IRIS の Web サーバ・ポートを指定すると共に、非 root コンテナレス・モードでの Web ゲートウェイ・ポートも指定します。既定値は 52773 です。

A.7.2 プロビジョニング・フェーズ

ICM では、ICM によってプロビジョニングされるインフラストラクチャ (PreExisting 導入の場合はユーザが指定したインフラストラクチャ) のホストへの root アクセスが必要な必須の構成はすべて、icm provision コマンドを使用してそのインフラストラクチャをプロビジョニングする前に完了していることが前提となっています。プロビジョニング・フェーズ中に、ICM はその要件が満たされているかどうかを確認します。満たされていない場合は、見つかった問題を示すメッセージと、その軽減方法の提案が表示されます。以下の例は、簡略化のために編集されています。

```
# icm provision
Configuring Acme-DATA-TEST-0001...
provision on Acme-DATA-TEST-0001 failed (attempt 1 of 1)...
Acme-DATA-TEST-0001 - Error: Thread exited with value 1
SSH operation failed
See /Samples/GCP/state/Acme-DATA-TEST/Acme-DATA-TEST-0001/ssh.err
  and /Samples/GCP/state/Acme-DATA-TEST/Acme-DATA-TEST-0001/ssh.out
```

```
Errors occurred during execution; aborting operation
To reprovision, specify -stateDir state

# cat /Samples/GCP/state/Acme-DATA-TEST/Acme-DATA-TEST-0001/ssh.err
20201203-21:19:16:320888393 Error: 2 nonroot issues found;
  please examine configNode.log, address the issues, and re-provision

# cat /Samples/GCP/state/Acme-DATA-TEST/Acme-DATA-TEST-0001/ssh.out
------------------------------------------------------------------------
Potential issue found: Cannot determine if requirement met for
  nonroot deployment: Configure iptables
# /sbin/iptables --delete INPUT --jump REJECT --reject-with icmp-host-prohibited
# /sbin/iptables --delete FORWARD --jump REJECT --reject-with icmp-host-prohibited
# /sbin/iptables-save
------------------------------------------------------------------------
Issue found: Requirement not met for nonroot deployment: IP forwarding
# echo "net.ipv4.ip_forward=1" | tee -a /etc/sysctl.conf
# sysctl -w net.ipv4.ip_forward=1
------------------------------------------------------------------------
Potential issue found: Cannot determine if requirement met for
  nonroot deployment on ubuntu: Retrieve apt-get updates
# apt-get update
------------------------------------------------------------------------
Issue found: Requirement not met for nonroot deployment on ubuntu:
  Install net-tools
# apt-get --allow-unauthenticated -y install net-tools
------------------------------------------------------------------------
Potential issue found: Cannot determine if requirement met for
  nonroot deployment on ubuntu: Clean apt-get cache
# apt-get -y clean all
------------------------------------------------------------------------
Potential issue found: Cannot determine if requirement met for
  nonroot deployment on GCP: Configure core dump
# systemctl disable apport; systemctl stop apport; sysctl -p;
  sysctl -w kernel.core_pattern='%e.%p.%h.%t.core'
# echo "%e.%p.%h.%t.core" > /proc/sys/kernel/core_pattern
```

このようにすると、関係するシステムでこれらの変更を root として直接実行できます（または、システムの VM テンプレートの更新として実行できます）。

注釈
- コマンドを個別に実行する必要はありません。すべてのコマンドを 1 回の操作でまとめて実行できます。
- これらのコマンドは、root として実行されるよう構成されています。一部のコマンドは sudo で動作するように形式を変更する必要があります。
- “考えられる問題”に対して提案されるコマンドの一部は、必須でないために正しく機能しない場合がありますが、これは正常です。

必要な変更を加えたら、コマンド icm provision -force を使用して再プロビジョニングします。これで、この付録で前述したように icm install コマンドを使用して導入を行う準備ができました。

B

ICM 導入環境の共有

重要　InterSystems IRIS リリース 2023.3 以降、InterSystems Cloud Manager (ICM) は非推奨になっています。今後のバージョンからは削除される予定です。

異なるシステムの異なるユーザが ICM を使用して同一の導入環境を管理または操作する必要のある状況はたくさんあります。例えば、1 人のユーザがインフラストラクチャのプロビジョニングを担当し、異なる場所の別のユーザがアプリケーションの導入とアップグレードを担当する場合があります。

しかしながら、ICM 導入環境は、入力ファイルによって定義されるため、いくつかの出力ファイルが生成されることになります。導入元である ICM コンテナ内のこれらの状態ファイルにアクセスできなければ、導入環境の管理や監視はどのユーザ (これらのファイルが失われた場合、元の導入担当者も含まれる) にとっても困難です。

このタスクを簡単にするために、ICM を分散管理モードで実行できます。このモードでは、ICM は導入の状態ファイルを Consul クラスタに保存するため、他の追加の ICM コンテナからもアクセスできるようになります。分散管理モードを使用しない場合、状態ファイルを手動で共有することもできます。

B.1 分散管理モードでの導入環境の共有

ICM の分散管理モードでは、Hashicorp の Consul サービス検出ツールを使用して、ネットワーク接続された任意の場所にいる複数のユーザに、単一の ICM 導入環境への管理アクセス権が付与されます。これは、複数の ICM コンテナを使用することで実行されます。各 ICM コンテナには、必要な状態ファイルを保存する 1 つ以上の Consul サーバでクラスタ化された Consul クライアントが 1 つずつ含まれます。

- 分散管理モードの概要
- 分散管理モードの構成
- 分散管理モードを使用した ICM のアップグレード

B.1.1 分散管理モードの概要

インフラストラクチャのプロビジョニングに使用する最初の ICM コンテナをプライマリ ICM コンテナ (または単に “プライマリ ICM”) と呼びます。このプロビジョニング・フェーズでは、プライマリ ICM は以下を実行します。

- 1 台、3 台、または 5 台の Consul サーバを CN ノードに導入します。これらのノードは、Consul クライアントと共にクラスタ化されます。
- icm provision コマンドの完了時に以下を実行します。
  - 導入の状態ファイル (指定内容については ″状態ファイル″ を参照) を Consul クラスタにプッシュします。

- 導入環境用にさらに ICM コンテナを作成するために docker run コマンドを出力します。

出力された docker run コマンドをユーザが実行すると、セカンダリ ICM コンテナ (“セカンダリ ICM”) が作成され、プロバイダに応じたディレクトリ (例えば、/Samples/GCP) でインタラクティブ・コンテナ・セッションが開始します。セカンダリ ICM では、ICM コマンドが開始するたびに、導入の状態ファイルが Consul クラスタから自動的にプルされるため、セカンダリ ICM では常に最新情報が保持されます。これによって、インフラストラクチャのプロビジョニングもプロビジョニング解除もできないことを除き、プライマリ ICM コンテナのまったくの複製であるコンテナが作成されます。その他の ICM コマンドはすべて有効です。

B.1.2 分散管理モードの構成

プライマリ ICM コンテナおよび Consul クラスタを作成するには、以下の操作を行います。

1. ConsulServers フィールドを defaults.json ファイルに追加して、以下のように Consul サーバの数を指定します。

   ```
   “ConsulServers”: “3”
   ```

   指定可能な値は、1、3、および 5 です。単一 Consul サーバは、単一障害点となるため、お勧めしません。5 台のサーバによるクラスタは、3 台のサーバによるクラスタより信頼性が高まりますが、コストも高くなります。

2. definitions.json ファイルに、CN ノード定義を追加します。このとき、ConsulServers フィールドの値以上の数の CN ノードを指定します。以下に例を示します。

   ```
   {
        "Role": "CN",
        "Count": "3",
        "StartCount": "7",
        "InstanceType": "t2.small"
   }
   ```

3. ICM コンテナで、プライマリ ICM の docker run コマンドに consul.sh スクリプトを追加します。

   ```
   docker run --name primary-icm --init -d -it --cap-add SYS_TIME
     intersystems/icm:latest-em consul.sh
   ```

プライマリ ICM コマンド行で、icm provision コマンドを発行すると、各 CN ノードがプロビジョニングされる際に、それらの各 CN ノードに Consul サーバが、指定したサーバ台数に達するまで導入されます。このコマンドが正常に完了すると、プライマリ ICM は状態ファイルを Consul クラスタにプッシュします。その出力にはセカンダリ ICM の作成コマンドが含まれます。プライマリ ICM で、インスタンス・ファイルを変更する可能性のあるコマンド (icm run や icm upgrade など) を次に発行すると、プライマリ ICM は新しいファイルを Consul クラスタにプッシュします。プライマリ ICM で icm unprovision コマンドを使用して導入のプロビジョニング解除を行うと、その状態ファイルは Consul クラスタから削除されます。

icm provision コマンドによる出力で提供されるセカンダリ ICM の icm run コマンドには暗号化キー (16 バイト、Base64 でエンコードされたもの) が含まれ、新しい ICM コンテナは Consul クラスタに参加することができます。以下に例を示します。

```
docker run --name icm --init -d -it --cap-add SYS_TIME intersystems/icm:latest-em
  consul.sh qQ6MPKCH1YzTb0j9Yst33w==
```

セカンダリ ICM 作成コマンドは、必要なだけ何回でも、導入へのネットワーク・アクセスがある任意の場所で使用できます。

プライマリ ICM コンテナとセカンダリ ICM コンテナの両方で、consul members コマンドを使用して、Consul クラスタに関する情報を表示できます。以下に例を示します。

```
/Samples/GCP # consul members
Node                                Address                Status  Type    Build  Protocol DC  Segment
consul-Acme-CN-TEST-0002.weave.local  104.196.151.243:8301  failed  server  1.1.0  2        dc1 <all>
consul-Acme-CN-TEST-0003.weave.local  35.196.254.13:8301    alive   server  1.1.0  2        dc1 <all>
consul-Acme-CN-TEST-0004.weave.local  35.196.128.118:8301   alive   server  1.1.0  2        dc1 <all>
3be7366b4495                        172.17.0.4:8301       alive   client  1.1.0  2        dc1 <default>
e0e87449a610                        172.17.0.3:8301       alive   client  1.1.0  2        dc1 <default>
```

以下に示したように、Consul コンテナは、icm ps コマンドの出力にも含まれます。

```
Samples/GCP # icm ps
Pulling from consul cluster...
CurrentWorkingDirectory: /Samples/GCP
...pulled from consul cluster
Machine            IP Address       Container           Status   Health   Image
-------            ----------       ---------           ------   ------   -----
Acme-DM-TEST-0001  35.227.32.29     weave               Up                weaveworks/weave:2.3.0
Acme-DM-TEST-0001  35.227.32.29     weavevolumes-2.3.0  Created           weaveworks/weaveexec:2.3.0
Acme-DM-TEST-0001  35.227.32.29     weavedb             Created           weaveworks/weavedb:2.3.0
Acme-CN-TEST-0004  35.196.128.118   consul              Up                consul:1.1.0
Acme-CN-TEST-0004  35.196.128.118   weave               Up                weaveworks/weave:2.3.0
Acme-CN-TEST-0004  35.196.128.118   weavevolumes-2.3.0  Created           weaveworks/weaveexec:2.3.0
Acme-CN-TEST-0004  35.196.128.118   weavedb             Created           weaveworks/weavedb:2.3.0
Acme-CN-TEST-0002  104.196.151.243  consul              Up                consul:1.1.0
Acme-CN-TEST-0002  104.196.151.243  weave               Up                weaveworks/weave:2.3.0
Acme-CN-TEST-0002  104.196.151.243  weavevolumes-2.3.0  Created           weaveworks/weaveexec:2.3.0
Acme-CN-TEST-0002  104.196.151.243  weavedb             Created           weaveworks/weavedb:2.3.0
Acme-CN-TEST-0003  35.196.254.13    consul              Up                consul:1.1.0
Acme-CN-TEST-0003  35.196.254.13    weave               Up                weaveworks/weave:2.3.0
Acme-CN-TEST-0003  35.196.254.13    weavevolumes-2.3.0  Created           weaveworks/weaveexec:2.3.0
Acme-CN-TEST-0003  35.196.254.13    weavedb             Created           weaveworks/weavedb:2.3.0
```

注釈　ICM コマンドには並行処理の制御が適用されないため、複数の ICM コンテナで競合するコマンドを同時に発行すると、一部のコマンドが失敗します。結果はタイミングに基づき、エラーが含まれる場合があります。例えば、異なるコンテナの 2 人のユーザが同時に icm rm -machine Acme-DM-TEST-0001 コマンドを発行したとします。一方のユーザには、以下が表示されます。

```
Removing container iris on Acme-DM-TEST-0001...
...removed container iris on Acme-DM-TEST-0001
```

他方のユーザには、以下が表示されます。

```
Removing container iris on Acme-DM-TEST-0001...
Error: No such container: iris
```

ただし、競合が存在しない場合、同じコマンドを同時にエラーなしに実行することができます。例えば、icm rm -machine Acme-DM-TEST-0001 コマンドや icm rm -container customsensors -machine Acme-DM-TEST-0001 コマンドです。

B.1.3 分散管理モードを使用した ICM のアップグレード

"分散管理モードの概要" で説明したように、分散管理モードでは導入の状態ファイルが Consul クラスタに保存されるので、これらのファイルを失うことなく、ICM コンテナを簡単にアップグレードできます。

最新バージョンを使用するという利点のほかに、インターシステムズのコンテナをアップグレードするときにも ICM のアップグレードが必要になります。ICM を起動するイメージと導入するインターシステムズのイメージのメジャー・バージョンが一致している必要があるためです。例えば、2022.1 バージョンの ICM を使用して 2022.2 バージョンの InterSystems IRIS を導入することはできません。そのため、インターシステムズのコンテナをアップグレードする前に ICM をアップグレードする必要があります。

分散管理モードで ICM コンテナをアップグレードするには、以下の手順を使用します。

1. プライマリ ICM コンテナによるプロビジョニングの終了時に提供されるセカンダリ ICM の icm run コマンドを使用して(”分散管理モードの構成” を参照)、アップグレード先の ICM イメージからセカンダリ ICM コンテナを作成します(異なる ICM イメージから作成されたプライマリ ICM コンテナとセカンダリ ICM コンテナは互換性があります)。
2. プライマリ ICM コンテナで、コマンド consul.sh show-master-token を発行して Consul トークンの値を取得します。
3. アップグレードされたセカンダリ ICM コンテナで、コマンド consul.sh convert-to-thick Consul_token を発行して、それをプライマリ ICM コンテナに変換します。
4. docker stop と docker rm を使用して、古いプライマリ ICM コンテナを停止して削除します。

現在の導入環境を管理している ICM コンテナをアップグレードする際にはこの方法が推奨されるので、分散管理を使用するかどうかに関係なく、このオプションを使用できるように、ICM を使用するたびにプライマリ ICM コンテナを作成することをお勧めします。

B.2 導入環境の手動による共有

ここでは、ICM 導入環境を手動で共有する方法について説明します。ICM による導入環境を他のユーザと共有したり、別の場所から導入環境にアクセスしたりできるように、導入環境の共有に必要な状態ファイル、コンテナの外部からこれらのファイルにアクセスする方法、およびこれらのファイルを維持する方法を示します。

- 状態ファイル
- 不変性の維持
- 状態ファイルの維持

B.2.1 状態ファイル

状態ファイルは現在の作業ディレクトリから読み書きされますが、カスタムの名前と場所を使用するようにこれらのファイルすべてをオーバーライドできます。入力ファイルは、以下のとおりです。

- defaults.json
- definitions.json

これらの構成ファイル内で参照されるセキュリティ・キー、InterSystems IRIS ライセンス、またはその他のファイルも、入力と見なす必要があります。

出力ファイルは、以下のとおりです。

- instances.json
- state/ ディレクトリ (-stateDir path でオーバーライドできます)

state/ の下にあるファイルのレイアウトは、以下のとおりです。

```
definition 0/
definition 1/
...
definition N/
```

それぞれの定義ディレクトリの下には、以下のファイルがあります。

- terraform.tfvars — Terraform 入力
- terraform.tfstate — Terraform 状態

さまざまなログ・ファイル、一時ファイル、およびその他のファイルもこの階層内に存在しますが、これらは導入環境の共有には必要ありません。

注釈　　プロバイダが PreExisting の場合、Terraform ファイルは生成されません。

B.2.2 不変性の維持

以下の理由から、ICM コンテナのローカル側では状態ファイルを生成しないことをお勧めします。

- 不変性が損なわれます。
- コンテナが削除/更新/置換されると、データが失われる可能性があります。
- ICM コンテナ内で構成ファイルを編集できる範囲は制限されています。
- コンテナの外部に状態ファイルをコピーする必要があり、この作業は面倒でエラーの原因になります。

より良い方法は、作業ディレクトリとして使用するディレクトリを ICM コンテナ内でホストからマウントすることです。このようにすれば、コンテナ内でのすべての変更を常にホスト上で取得できます。このためには、以下のように、ICM コンテナを初めて作成するときに Docker --volume オプションを使用します。

```
$ docker run --name icm --init -d -it -cap-add SYS_TIME --volume <host_path>:<container_path> <image>
```

全体としては、以下の手順を実行します。

1. ホスト上の入力ファイルを host_path 内でステージングします。
2. ICM コンテナを作成し、起動して、コンテナにアタッチします。
3. container_path に移動します。
4. ICM コマンドを発行します。
5. ICM コンテナを終了するか、アタッチを解除します。

これで、状態ファイル（入力と出力の両方）が host_path に格納されます。この方法の例については、"ICM の起動" のサンプル・スクリプトを参照してください。

B.2.3 状態ファイルの維持

状態ファイルを維持し、他のユーザと共有するには、以下のような方法があります。

- tar/gzip を作成する

  作成したアーカイブを電子メールで送信したり、FTP サイトに配置したり、USB メモリに書き込むことができます。
- 他のユーザがリストアできる場所にバックアップを作成する

  ホスト上の状態ファイルのパスをバックアップ・サービスに登録します。
- 組織内の他のユーザがアクセスできるディスク・ボリュームをマウントする

  例えば、状態ファイルのパスを Samba マウントにできます。
- クラウドにバックアップされるディスクの場所を指定する

  Dropbox、Google Drive、OneDrive などのサービスを使用できます。
- ドキュメント・データベースに保管する

  クラウド・ベースまたはオンプレミスのデータベースを使用できます。

後に挙げた 3 つの方法の利点は、他のユーザが導入環境を変更できることです。ただし、複数の ICM コンテナから一度に発行される同時操作は ICM によるサポート対象外なので、排他的な読み取り/書き込みアクセスを保証するポリシーを施行する必要があります。

C

ICM によるスクリプト作成

重要　　InterSystems IRIS リリース 2023.3 以降、InterSystems Cloud Manager (ICM) は非推奨になっています。今後のバージョンからは削除される予定です。

この付録では、スクリプトから一連の ICM コマンドを発行する方法と、導入環境全体にわたってコンテナとサービスを指定し、調整する方法を説明します。

C.1 ICM の終了ステータス

ICM は、それぞれのコマンドの後に UNIX 終了ステータスを設定します。これは、指定した ICM コマンドが正常に実行されたかどうか簡単に判断する手段です。以下の例では、それぞれのコマンドの後に特殊変数 **$?** を検査しています。

```
# icm inventory
Machine                 IP Address    DNS Name                        Region   Zone
-------                 ----------    --------                        ------   ----
Acme-DATA-TEST-0001 54.191.233.2  ec2-54-191-233-2.amazonaws.com  us-east1 b
Acme-DATA-TEST-0002 54.202.223.57 ec2-54-202-223-57.amazonaws.com us-east1 b
Acme-DATA-TEST-0003 54.202.223.58 ec2-54-202-223-58.amazonaws.com us-east1 b
# echo $?
0

# icm publish
Unrecognized goal: 'publish' (try "icm -help")
# echo $?
1

# icm ps -role QM
Unrecognized role 'QM'
# echo $?
1

# icm session
Error: Interactive commands cannot match more than one instance
# echo $?
1
```

C.2 ICM ロギング

ICM は、出力をファイル (既定では icm.log) とコンソールに記録します。すべての出力がログ・ファイルに送られますが、そのコンソール出力を取得して stdout と stderr に分割できます。以下の例は、エラーを出さずに完了しています。

```
# icm inventory > std.out 2> std.err
# cat std.out
Machine             IP Address    DNS Name                          Region   Zone
-------             ----------    --------                          ------   ----
Acme-DATA-TEST-0001 54.191.233.2  ec2-54-191-233-2.amazonaws.com  us-east1 b
Acme-DATA-TEST-0002 54.202.223.57 ec2-54-202-223-57.amazonaws.com us-east1 b
Acme-DATA-TEST-0003 54.202.223.58 ec2-54-202-223-58.amazonaws.com us-east1 b
# cat std.err
```

以下の例は、エラー出力を含んでいます。

```
# icm publish > std.out 2> std.err
# cat std.out
# cat std.err
Unrecognized goal: 'publish' (try "icm -help")
```

C.3 リモート・スクリプト呼び出し

コマンドを組み合わせて使用して、ホストまたはコンテナにスクリプトをコピーし、リモート側からこれらのスクリプトを呼び出すことができます。以下の例では、エクスポートした InterSystems IRIS ルーチンを InterSystems IRIS クラスタにコピーした後、コンパイルして実行します。

```
# icm scp -localPath Routine1.xml -remotePath /tmp/
# icm session -command 'Do ##class(%SYSTEM.OBJ).Load("/tmp/Routine1.xml", "c-d")'
# icm session -command 'Do ^Routine1'
```

以下の例では、ホストにシェル・スクリプトをコピーし、アクセス権を変更して、実行します。

```
# icm scp -localPath script1.sh -remotePath /tmp/
# icm ssh -command 'sudo chmod a+x /tmp/script1.sh'
# icm ssh -command '/tmp/script1.sh abc 123'
```

以下の例でも実行することは同じですが、カスタムまたはサードパーティのコンテナ内で行います。

```
# icm cp -localPath script2.sh -remotePath /tmp/ -container gracie
# icm exec -command 'chmod a+x /tmp/script2.sh' -container gracie
# icm exec -command '/tmp/script2.sh abc 123' -container gracie
```

C.4 JSON モードの使用

スクリプトで、クラスタの状態に関する情報を ICM から収集する必要が生じることがあります。次に例を示します。

- InterSystems IRIS データ・サーバの IP アドレスは?
- カスタム/サードパーティ・コンテナのステータスは?

ICM の目視可読形式の出力を解析することは困難で、誤りが生じやすくなります。より確実なソリューションは、json オプションを使用して JSON 形式で出力を生成するように ICM に指示することです。出力は、現在の作業ディレクトリ内のファイル responsejson に書き込まれます。

- 正常な出力

- 異常な出力

C.4.1 正常な出力

ほとんどの ICM コマンドは、正常終了時に出力を生成しません。この場合、終了値は 0 で、stderr に出力は書き込まれず、JSON は空の配列です。

```
# icm exec -command "ls /" -json
# cat response.json
[]
```

正常終了時に有用な出力を生成する ICM コマンドについて、以下に詳しく説明します。フィールドの順序は保証されません。

C.4.1.1 icm provision

icm provision 出力の形式は、プロビジョニング時に使用される入力 (つまり、構成) ファイルと出力 (つまり状態) ファイルを表す名前と値のペアが含まれるオブジェクトです。次の例で示しているように、対応するコマンド行引数に一致する名前は、defaults、definitions、instances、および stateDir です。

```
# icm provision -json
...
Machine             IP Address    DNS Name                         Region   Zone
-------             ----------    --------                         ------   ----
Acme-DATA-TEST-0001 54.191.233.2  ec2-54-191-233-2.amazonaws.com   us-east1 b
Acme-DATA-TEST-0002 54.202.223.57 ec2-54-202-223-57.amazonaws.com  us-east1 b
Acme-DATA-TEST-0003 54.202.223.58 ec2-54-202-223-58.amazonaws.com  us-east1 b
To destroy: icm unprovision [-cleanUp] [-force]

# cat response.json
{
  "defaults" : "defaults.json",
  "definitions" : "definitions.json",
  "instances" : "instances.json",
  "stateDir" : "/Samples/AWS/state/"
}
```

C.4.1.2 icm inventory

icm inventory コマンドの形式は、プロビジョニングされたそれぞれのインスタンスに対応する要素を含む配列であり、それぞれの要素は名前と値のペア MachineName、Role、IPAddress、および DNSName からなるリストを含みます。例を以下に示します。

```
# icm inventory -json
Machine             IP Address    DNS Name                         Region   Zone
-------             ----------    --------                         ------   ----
Acme-DATA-TEST-0001 54.191.233.2  ec2-54-191-233-2.amazonaws.com   us-east1 b
Acme-DATA-TEST-0002 54.202.223.57 ec2-54-202-223-57.amazonaws.com  us-east1 b
Acme-DATA-TEST-0003 54.202.223.58 ec2-54-202-223-58.amazonaws.com  us-east1 b

# cat response.json
[
  {
    “Provider”: “AWS”,
    "MachineName":"Acme-DATA-TEST-0001",
    "Role":"DATA",
    "IPAddress":"54.191.233.2",
    "DNSName":"54_191_233_2.amazonaws.com",
    “Region”: “us-east-1”,
    “Zone”: “us-east-1b”
  },
  {
    “Provider”: “AWS”,
    "MachineName ":"Acme-DATA-TEST-0002",
    "Role":"DATA",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com,
    “Region”: “us-east-1”,
    “Zone”: “us-east-1b””
  },
```

```
  {
    “Provider”: “AWS”,
    "MachineName":"Acme-DATA-TEST-0003",
    "Role":"DATA",
    "IPAddress":"54202.223.58",
    "DNSName":"54_202_223_58.amazonaws.com,
    “Region”: “us-east-1”,
    “Zone”: “us-east-1b”"
  }
]
```

C.4.1.3 icm ps

コンテナ・モードでは、icm ps コマンドの出力は、それぞれのコンテナに対応する要素を含む配列であり、それぞれの要素は名前と値のペア MachineName、Role、IPAddress、DNSName、Container、DockerImage、Status、および MirrorStatus (該当する場合) からなるリストを含みます。

```
# icm ps -container iris -json
Machine              IP Address     Container  Status  Health  Image
-------              ----------     ---------  ------  -----   -----
Acme-DATA-TEST-0001  54.191.233.2   iris       Up      healthy isc/iris:latest-em
Acme-DATA-TEST-0002  54.202.223.57  iris       Up      healthy isc/iris:latest-em
Acme-DATA-TEST-0003  54.202.223.58  iris       Up      healthy isc/iris:latest-em
# cat response.json
[
  {
    "MachineName":"Acme-DATA-TEST-0001",
    "Role":"DATA",
    "IPAddress":"54.191.233.2",
    "DNSName":"54_191_233_2.amazonaws.com"
    "Container":"iris",
    "DockerImage":"isc/iris:latest-em",
    "Status":"Up",
    "Health":"healthy"
  },
  {
    "MachineName ":"Acme-DATA-TEST-0002",
    "Role":"DATA",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
    "Container":"iris",
    "DockerImage":"isc/iris:latest-em",
    "Status":"Up"",
    "Health":"healthy"
  },
  {
    "MachineName ":"Acme-DATA-TEST-0003",
    "Role":"DATA",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
    "Container":"iris",
    "DockerImage":"isc/iris:latest-em",
    "Status":"Up",
    "Health":"healthy"
  },
]
```

コンテナレス・モードでは、icm ps の出力フィールドは、MachineName、Role、IPAddress、DNSName、ISCInstance (インターシステムズ提供のコンテナ内では常に **IRIS**)、Kit、Status、および MirrorStatus (該当する場合) です。

```
# icm ps -json
Machine             IP Address    Instance Kit            Status   Health
-------             ----------    -------- ---            ------   ------
Acme-DATA-TEST-0001 54.67.2.117   IRIS     2017.3.0.392.0 running  ok
Acme-DATA-TEST-0002 54.153.96.236 IRIS     2017.3.0.392.0 running  ok
Acme-DATA-TEST-0002 54.153.90.66  IRIS     2017.3.0.392.0 running  ok

# cat response.json
[
  {
    "MachineName":"Acme-DATA-TEST-0001",
    "Role":"DATA",
    "IPAddress":"54.191.233.2",
    "DNSName":"54_191_233_2.amazonaws.com"
    "ISCInstance":"IRIS",
    "Kit":"2017.3.0.392.0",
    "Status":"running",
    "Health":"ok"
  },
```

```
  {
    "MachineName ":"Acme-DATA-TEST-0002",
    "Role":"DATA",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
    "ISCInstance":"IRIS",
    "Kit":"2017.3.0.392.0",
    "Status":"running"",
    "Health":"ok"
  },
  {
    "MachineName ":"Acme-DATA-TEST-0003",
    "Role":"DATA",
    "IPAddress":"54.202.223.57",
    "DNSName":"54_202_223_57.amazonaws.com"
    "ISCInstance":"IRIS",
    "Kit":"2017.3.0.392.0",
    "Status":"running"",
    "Health":"ok"
  }
]
```

C.4.2 異常な出力

エラーが発生したときの JSON 出力の形式は、エラーが ICM アプリケーションのローカル側で発生したか、ホストまたはインスタンス上のターゲット・アプリケーションから発生したかによって異なります。

C.4.2.1 ローカル・エラー

ICM コマンドにエラーが発生した場合、JSON には以下のように、エラーを説明するオブジェクトと、名前と値のペアが含まれます。

```
# icm ps -role QM -json
Unrecognized role 'QM'

# cat response.json
{
    "error": "Unrecognized role 'QM'"
}
```

C.4.2.2 リモート・エラー

リモート・エラーは、以下のうち 1 つ以上が当てはまるときに発生したと見なされます。

- ゼロ以外の終了ステータス
- stderr への出力

リモート・エラーが発生すると、JSON は名前と値のペアを含むオブジェクトの配列になります。名前はターゲット・マシンのものに対応し、値は以下の 1 つ以上を含む名前と値のペアのリストを含む別のオブジェクトです。

- error : 発生した問題の説明 (例外のテキストの大部分)
- file : 問題に関する詳細を含むファイル
- exitValue : 基礎のプロセスの (ゼロ以外の) 終了値

次に例を示します。

```
# icm ssh -command "ls file.txt" -json
Executing command 'ls file.txt' on host Acme-DATA-TEST-0001...
ls: cannot access file.txt: No such file or directory
Error: See tmp/DATA-TEST/DATA-TEST-0001/ssh.err
Errors occurred during execution; aborting operation

# cat response.json
[
  {
    "Acme-DATA-TEST-0001": {
      "file": "tmp/DATA-TEST/DATA-TEST-0001/ssh.err"
    }
  }
]

# cat tmp/DATA-TEST/DATA-TEST-0001/ssh.err
ls: cannot access file.txt: No such file or directory
```

D

カスタム・コンテナおよびサードパーティ・コンテナでの ICM の使用

重要　InterSystems IRIS リリース 2023.3 以降、InterSystems Cloud Manager (ICM) は非推奨になっています。今後のバージョンからは削除される予定です。

この付録では、ICM を使用してユーザとサードパーティのコンテナを導入する方法について説明します。説明は、Docker イメージが ICM からアクセス可能なリポジトリに存在することを前提としています。他のコンテナおよびサービス (InterSystems IRIS を含む) と通信するようにコンテナを構成する方法については、"ICM によるスクリプト作成" を参照してください。

D.1 コンテナ名の指定

ホスト上で実行するコンテナには、それぞれ一意の名前を付ける必要があります。icm run を使用してコンテナを導入する際に、-container オプションを使用してコンテナに名前を付けることができます。

```
# icm run -container gracie -image docker/whalesay
```

icm ps の出力で、その名前が反映されていることを確認できます。

```
# icm ps
Machine           IP Address   Container Status     Health  Image
-------           ----------   --------- ------     ------  -----
Acme-DM-TEST-0001 172.16.110.9 gracie    Restarting         docker/whalesay
```

注釈　-container オプションが指定されない場合は、既定のコンテナ名 **iris** が使用されます。この名前は予約されており、インターシステムズが提供する InterSystems IRIS イメージから導入されたコンテナにのみ使用する必要があります。

D.2 既定のコマンドのオーバーライド

コンテナの既定のコマンドをオーバーライドする場合は、-command を使用して行うことができます。例えば、docker/whalesay イメージが既定でコマンド /bin/bash を実行するとします。

```
# icm docker -command "ps -a"

CONTAINER ID  IMAGE             COMMAND      CREATED     STATUS      NAMES
17f4ece54c2f  docker/whalesay  "/bin/bash"  4 days ago  Restarting  gracie
```

コンテナが実行するコマンドを別のものにする場合は (例えば pwd)、以下のようにコンテナを導入できます。

```
# icm run -container gracie -image docker/whalesay command pwd
```

以下のように Docker ログを調べて、コマンドが正常に実行されたことを確認できます。

```
# icm docker -command "logs gracie"
/cowsay
```

D.3 Docker オプションの使用

ICM によって明示的に提供されていない Docker オプションまたはオーバーライドがコンテナに必要になる場合があります。こういったオプションやオーバーライドは、-options オプションを使用して、追加できます。このセクションでは、より一般的なユース・ケースの例をいくつか示します。Docker オプションの詳細は、https://docs.docker.com/engine/reference/run/ を参照してください。

- 再起動
- 特権
- 環境変数
- マウント・ボリューム
- ポート

D.3.1 再起動

デフォルトでは、ICM はオプション --restart unless-stopped を指定してコンテナを導入します。つまり、コンテナが icm stop コマンド以外の理由で (コンテナの終了、Docker の再起動など) 実行境界を越えた場合、Docker はコンテナの実行を試み続けます。ただし、場合によっては、コンテナを 1 回限り実行して終了したままにする必要があります。この場合、以下のように再起動を抑止できます。

```
# icm run -container gracie -image docker/whalesay -options "--restart no"
# icm ps
Machine           IP Address   Container Status     Health   Image
-------           ---------    --------  -----      ------   -----
Acme-DM-TEST-0001 172.16.110.9 gracie    Exited (0)          docker/whalesay
```

D.3.2 特権

一部のコンテナを実行するには、追加の特権が必要です。また、既定の特権を削除することもできます。例を以下に示します。

```
# icm run -container sensors -image hello-world -options "--privileged"
# icm run -container fred -image hello-world -options "--cap-add SYS_TIME"
# icm run -container fred -image hello-world -options "--cap-drop MKNOD"
```

D.3.3 環境変数

Docker オプション --env を使用して、コンテナに環境変数を渡すことができます。これらの変数は、bash export コマンドと同様の方法によってコンテナ内部で設定されます。

```
# icm run container fred image hello-world options "--env TERM=vt100"
```

D.3.4 マウント・ボリューム

コンテナがホスト・マシン上のファイルにアクセスする必要がある場合は、Docker --volume オプションを使用して、コンテナ内にマウント・ポイントを作成できます。例えば、以下のように指定します。

```
# icm run container fred image hello-world options "--volume /dev2:/dev2"
```

これにより、ホスト上のディレクトリ /dev2 の内容が、コンテナ内のマウント・ポイント /dev2 で使用可能になります。

```
# icm ssh -command "touch /dev2/example.txt"  // on the host
# icm exec -command "ls /dev2"                // in the container
example.txt
```

D.3.5 ポート

Docker オプション --publish を使用して、コンテナ内のポートをホストにマップできます。

```
# icm run -container fred -image hello-world -options "--publish 80:8080"
# icm run -container fred -image hello-world -options "--publish-all"
```

外部からポートにアクセスする必要がある場合は、ホスト上の対応するポートを開く必要があります。これを行うには、以下のようにいくつかの方法があります。

- Terraform テンプレート・ファイル infrastructure.tf を直接編集する。
- icm ssh コマンドを使用してホストにコマンドを発行する。
- クラウド・プロバイダのコンソールでセキュリティ設定を変更する。

また、同じホスト上で別のコンテナまたはサービスにマップされたポートと衝突しないようにする必要があります。最後に、オーバーレイ・ネットワークのタイプが host である場合、--publish はコンテナに影響を及ぼさないことに注意してください。

次の例では、AWS 用の Terraform テンプレートを変更して、ポート 563 (NNTP over TLS) 経由で着信する TCP 通信を許可します。

- ファイル : /ICM/etc/Terraform/AWS/VPC/infrastructure.tf
- リソース : aws_security_group

- *ルール*：

```
ingress {
  from_port = 563
  to_port = 563
  protocol = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}
```

E

既存のクラスタへの導入

重要　InterSystems IRIS リリース 2023.3 以降、InterSystems Cloud Manager (ICM) は非推奨になっています。今後のバージョンからは削除される予定です。

ICM には、コンテナを導入するクラウドまたは仮想ホスト・ノードまたは物理サーバを独自に割り振るオプションが用意されています。プロビジョニング・フェーズには割り振りと構成のサブフェーズが通常は含まれますが、Provider フィールドが PreExisting に設定されている場合、ICM は割り振りフェーズをバイパスして構成に直接進みます。既存のクラスタに対してプロビジョニング解除のフェーズはありません。

以下のセクションでは、プロバイダ PreExisting を使用して既存のインフラストラクチャに導入する場合の要件について説明します。

重要　ICM は、それが実行されているホストにコンテナを導入することはできません。ICM にはそのホストの IP アドレスを特定する手段がないので、ユーザの責任で PreExisting 導入のホスト・ノードとして ICM ホストを指定しないようにしてください。

E.1 SSH

ICM を使用するには、SSH がインストールされていて、SSH デーモンが実行されている必要があります。

また、既定値ファイルの SSHUser フィールドで非 root アカウントを指定する必要があります。このアカウントには以下のような特性が必要です。

- パスワードを要求せずに sudo アクセスを提供する必要があります。このアクセスを可能にするには、/etc/sudoers.d/ 内でファイルを作成するか、またはファイルを変更して、以下の行を含めます。

  ```
  <accountname> ALL=(ALL) NOPASSWD:ALL
  ```

  パスワード・ログインを完全に禁止するには、SSHOnly パラメータを true に設定します。これにより、ICM はパスワードを使用してログインできなくなるので、SSH 公開鍵 (SSHPublicKey フィールドによって指定される) を各ノードでステージングする必要があります。

- ホーム・ディレクトリが /home 以外の場所にある場合は、既定値ファイルの Home フィールドにその場所を指定する必要があります。例を以下に示します。

  ```
  "Home": "/users/"
  ```

  ノード間で共有されるネットワーク・ディレクトリ (例えば、/nethome) をホーム・ディレクトリに指定すると、構成ファイルが別の構成ファイルを上書きするため、このように指定してはなりません。

ICM は、SSH 鍵またはパスワード・ログインを使用して、SSHUser としてログインできます。パスワード・ログインが有効になっている場合でも、ICM はまず SSH を使用したログインを試行します。

SSH 鍵を使用してマシンを構成した場合、**SSHPublicKey** フィールドと **SSHPrivateKey** フィールドを使用して、構成ファイルで SSH 公開/秘密鍵ペアを指定する必要があります。

構成フェーズ中に、ICM は SSH ログインを構成し、パスワード・ログインを既定では無効にします。パスワード・ログインを無効にしないようにする場合は、SSHUser アカウントのホーム・ディレクトリ内で以下の標識ファイルの touch を行うことができます。

```
mkdir -p ICM
touch ICM/disable_password_login.done
```

パスワードを使用してマシンを構成していた場合は、構成ファイル内の **SSHPassword** フィールドを使用してパスワードを指定します。ICM は、これらの資格情報が安全でないものと見なします。

パスワード・ログインを有効にして **SSHPassword** フィールドを指定しても、ICM が SSH 経由で構成後の操作すべてを実行できることは必要であり、この要件がなくなることはありません。

E.2 ポート

ローカル・セキュリティ・ポリシーとの競合を避けるため、またオペレーティング・システム間で差異があるために、ICM がポートのオープンを試行することはありません。以下の表に、さまざまな ICM 機能を使用するために開く必要がある既定のポートを示します。"ポートおよびプロトコルのパラメータ" に記載されているように、ポートは構成可能です。以下に例を示します。

```
"SuperServerPort": "51777"
```

以下に示すようにこれらの 1 つ以上のフィールドを既定値から変更する場合は、指定するポートが開いていることを確認する必要があります。

ポート	プロトコル	サービス	留意事項
22	tcp	SSH	必須項目。
2376	tcp	Docker (TLS モード)	必須項目。
80	tcp	Web	ロール WS (Web サーバ) のノード上のパブリック Apache Web サーバにアクセスするために必要。
53	tcp udp	DNS	Weave DNS に必要。
6783 6783 6784	tcp udp	Weave Net	Overlay=Weave に必要 (すべてのプロバイダの既定値)。
1972	tcp	InterSystems IRIS スーパーサーバ	必須項目。SuperServerPort フィールドを使用して、別のポートを指定できます。
52773	tcp	InterSystems IRIS Web サーバ	必須項目。WebServerPort フィールドを使用して、別のポートを指定できます。
2188	tcp	InterSystems IRIS ISCAgent	ミラーリングに必要。ISCAgentPort フィールドを使用して、別のポートを指定できます。
4002	tcp	InterSystems IRIS ライセンス・サーバ	必須。注：LicenseServerPort フィールドを使用して、別のポートを指定できます。

E.3 ストレージ・ボリューム

"ICM によってマウントされるストレージ・ボリューム" で説明したように、ICM は、InterSystems IRIS および Docker で使用するために提供するストレージ・ボリュームを、"デバイス名パラメータ" に示すフィールドで指定した名前を使用して /dev にマウントします。これらのデバイスの既定のマウント・ポイントを変更するには、対応するマウント・ポイント・パラメータで別の場所を指定します。

InterSystems IRIS で使用するために既存のインフラストラクチャ上に用意したボリュームを ICM でマウントするには、マウント・ポイント・パラメータでディレクトリを指定して、対応するデバイス名パラメータの 2 つの値のいずれかを使用できます。これを以下に示します。

- existing (既定値) – マウント・ポイント・パラメータで指定したディレクトリが存在する場合、ICM はそのディレクトリを対象のストレージ・ボリュームとして使用します。例えば、DataDeviceName の値が existing であるか、値が指定されておらず、DataMountPoint の値が /mnt/data である場合、ICM は /mnt/data を InterSystems IRIS インスタンスのデータ・ボリュームとしてマウントします。マウント・ポイント・パラメータで指定したディレクトリが存在しない場合、データ・ボリュームはマウントされません。
- create – existing と同じですが、次の点が異なります。マウント・ポイント・パラメータで指定したディレクトリが存在しない場合、ICM はそのディレクトリの作成を試行し、正常に作成できたら、それを対象のストレージ・ボリュームとしてマウントします。ディレクトリを作成できなかった場合、ストレージ・ボリュームはマウントされません。

E.4 PreExisting の定義ファイル

PreExisting とその他のプロバイダとの主な違いは、定義ファイルの内容です。この定義ファイルには、ノードごとに必ず 1 つずつのエントリが含まれます (ロールごとに 1 つのエントリがあってそのロールのノード数を指定する Count フィールドが含まれる定義ファイルとは異なります)。それぞれのノードは、IPAddress フィールドを使用して IP アドレスによって識別されます。