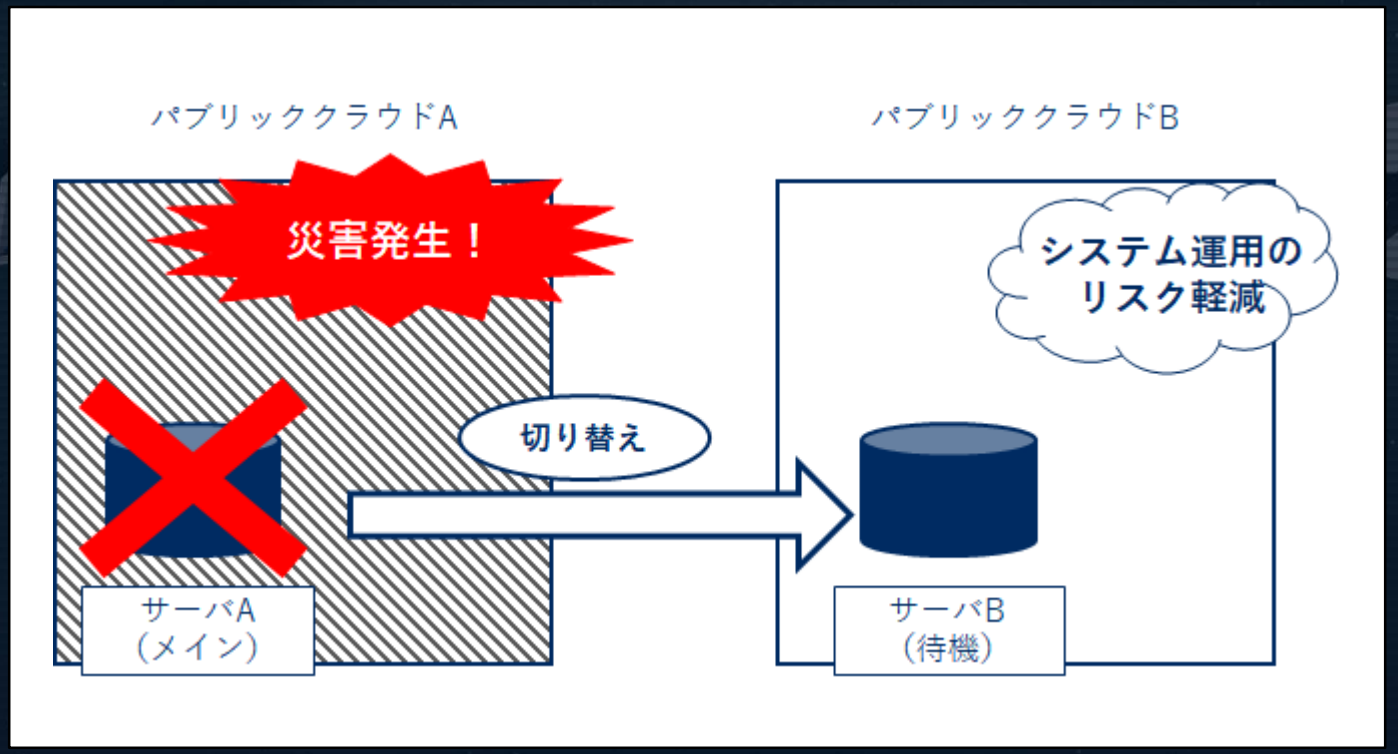
Exastro × Terraform × Ansibleで実現する マルチクラウドシステムプロビジョニング

アジェンダ

1.マルチクラウドとは？
2.Exastro・Terraformについて
3.デモ

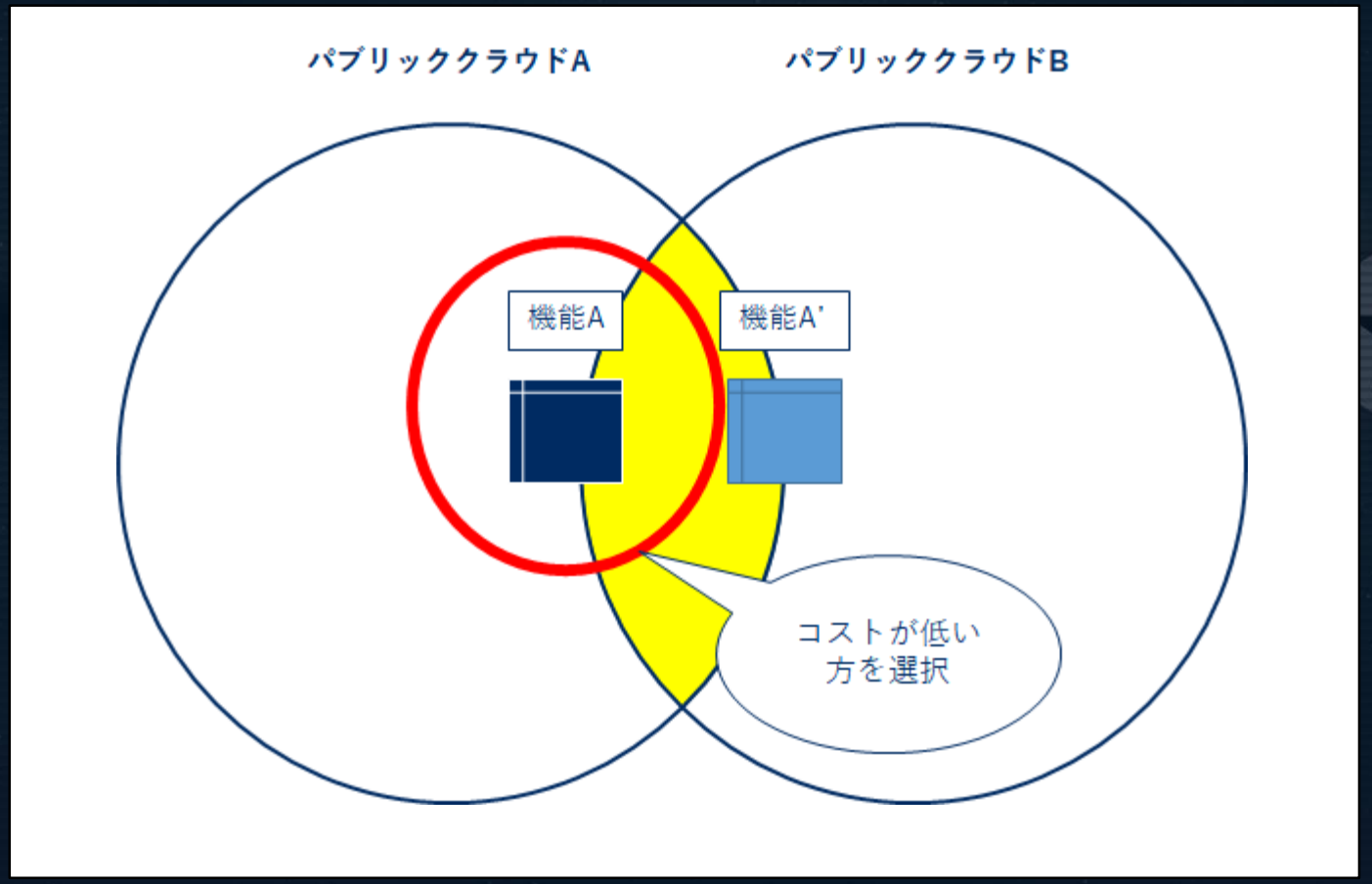
マルチクラウド

“複数のパブリッククラウドを使い分ける利用形態。データ通信量の分散、データ消失・システムダウンのリスク分散などの利点がある。 ” 引用：「デジタル大辞泉」



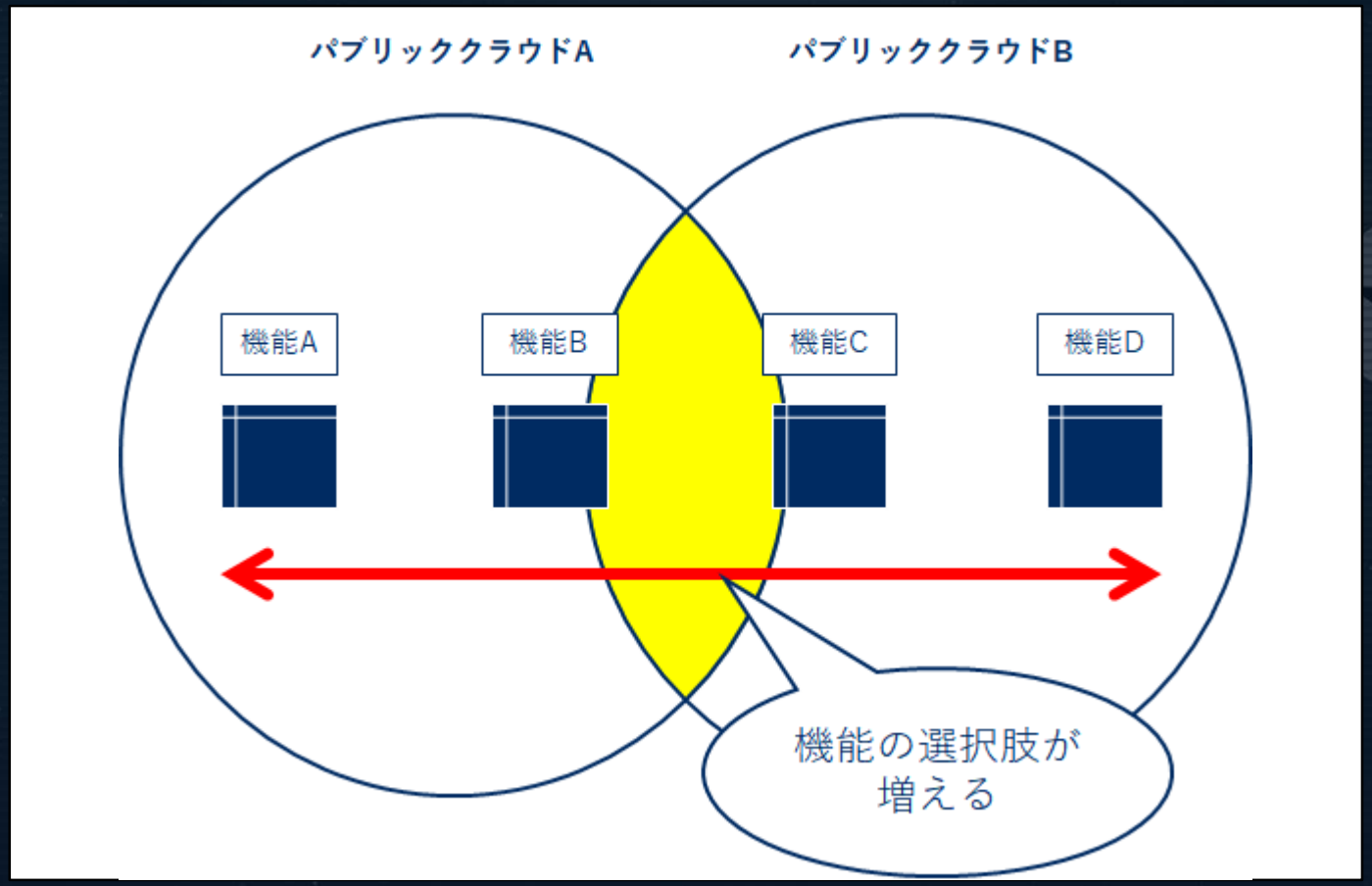
マルチクラウドへのさらなる期待 その１

パブリッククラウドの利用において、“コスト”は非常に重要な観点。
コストが安い機能の選択により、削減効果を見込める可能性あり。



マルチクラウドへのさらなる期待 その２

それぞれのパブリッククラウドごとに用意されている固有の機能を利用し、システムの構築を有利・スピーディに進めることができる。



マルチクラウドにおける課題とは？

マルチクラウドは以下のような問題を抱えている

✓エンジニアのスキル問題(※1)

✓導入までの遅れ(※2)

✓クラウドの管理や可視化(※3)

(※1) https://www.denodo.com/ja/document/whitepaper/denodo-global-cloud-survey-2020

(※2) https://japan.zdnet.com/article/35163287/

(※3) https://techtarget.itmedia.co.jp/tt/news/1908/30/news05.html

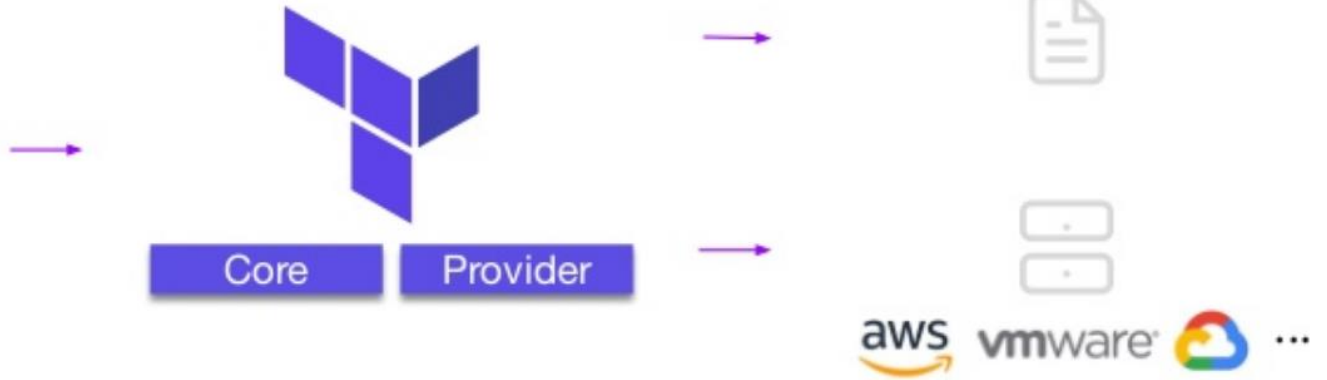
マルチクラウドの標準化・自動化・構成管理が必要とされている

マルチクラウドの構築に効果的なツールとは？

Terraform: Infrastructure as Code

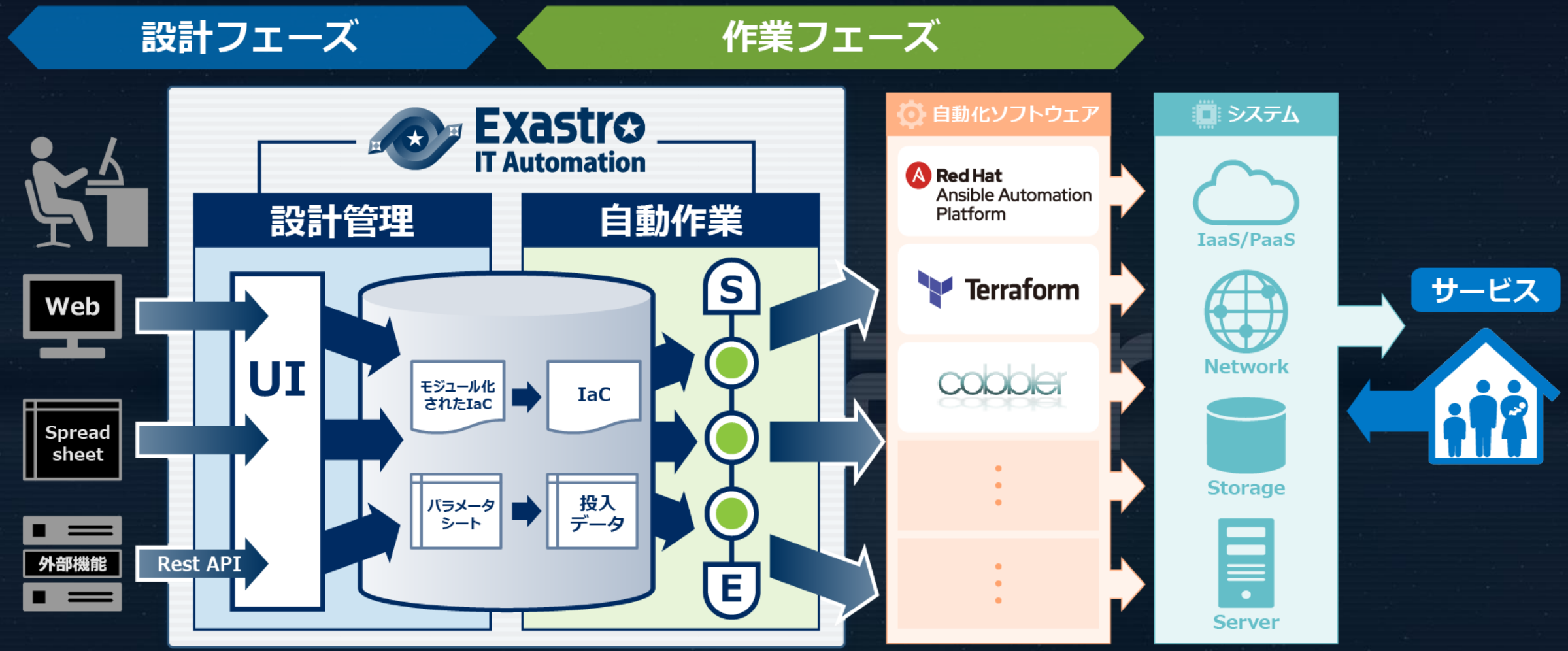
tfconfig

```
resource "vsphere_virtual_machine" "vm" {
  name             = "terraform-test"
  resource_pool_id = "*******"
  datastore_id     = "*******"
  num_cpus = 2
  memory   = 1024

  network_interface {
    network_id = "*******"
  }

  disk {
    label = "disk0"
    size  = 20
     }}
```

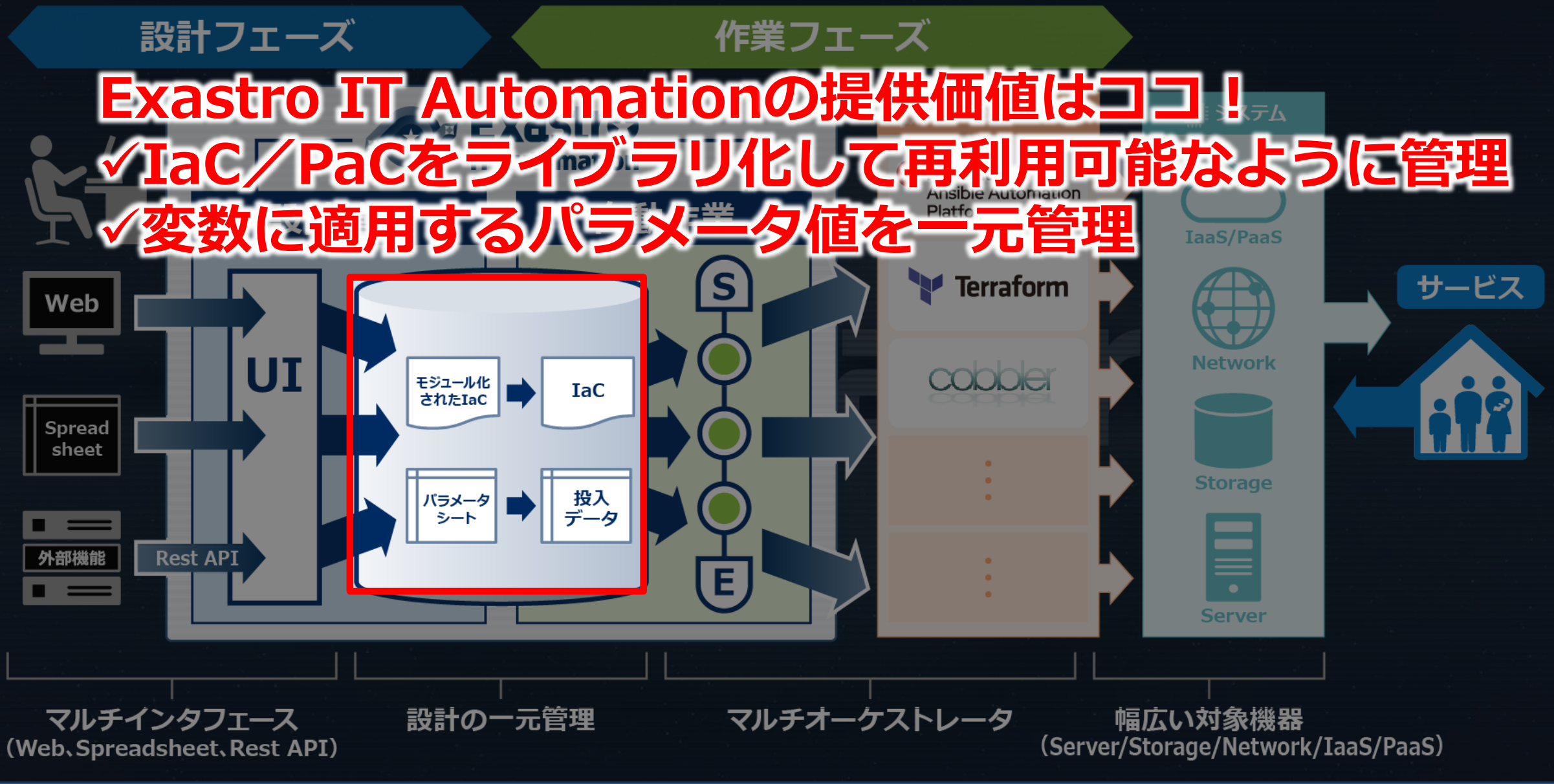


https://www.slideshare.net/hashicorpjp/terraform-enterprise
https://www.terraform.io/

Exastro IT Automation：システム情報をデジタル管理するためのフレームワーク



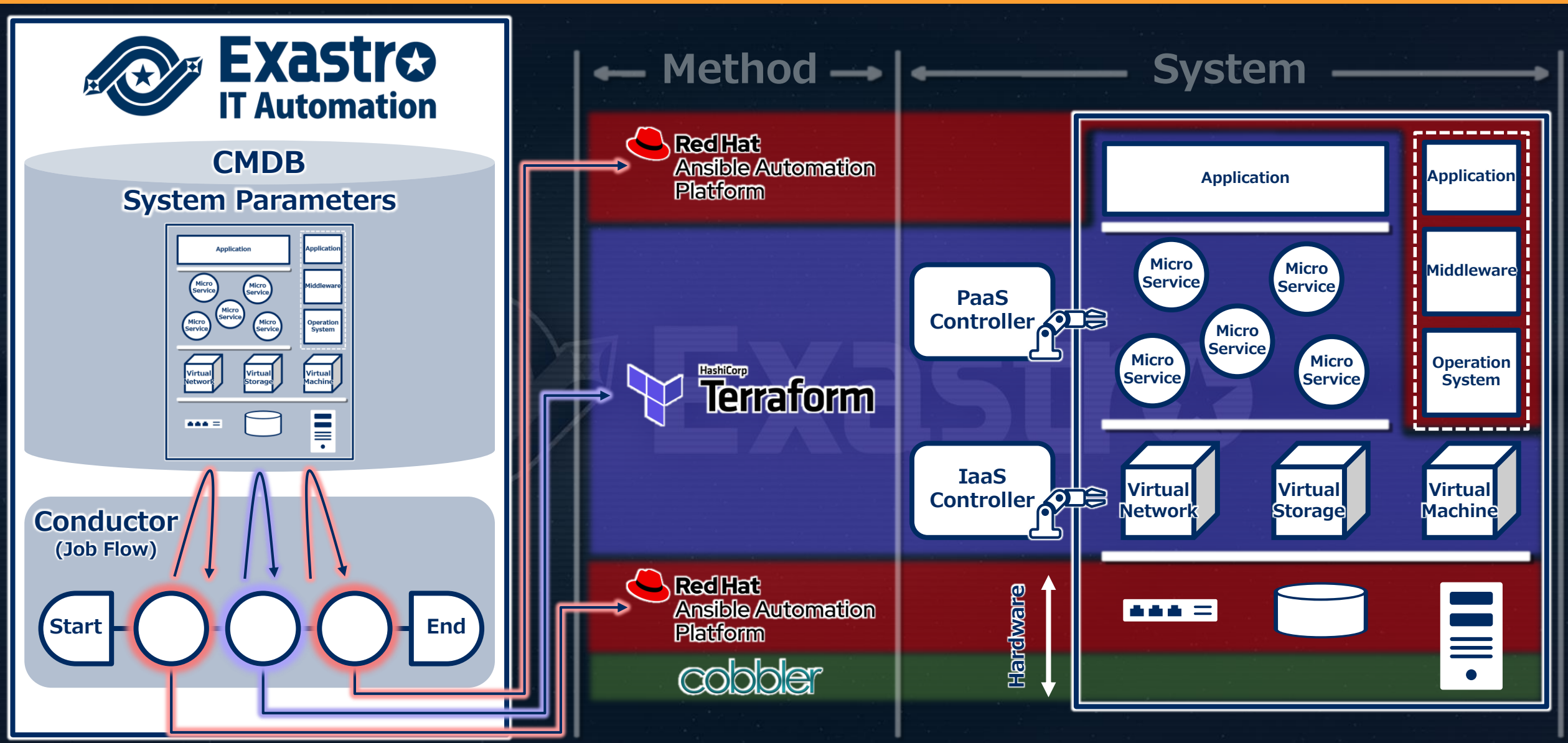
Exastro IT Automation：システム情報をデジタル管理するためのフレームワーク



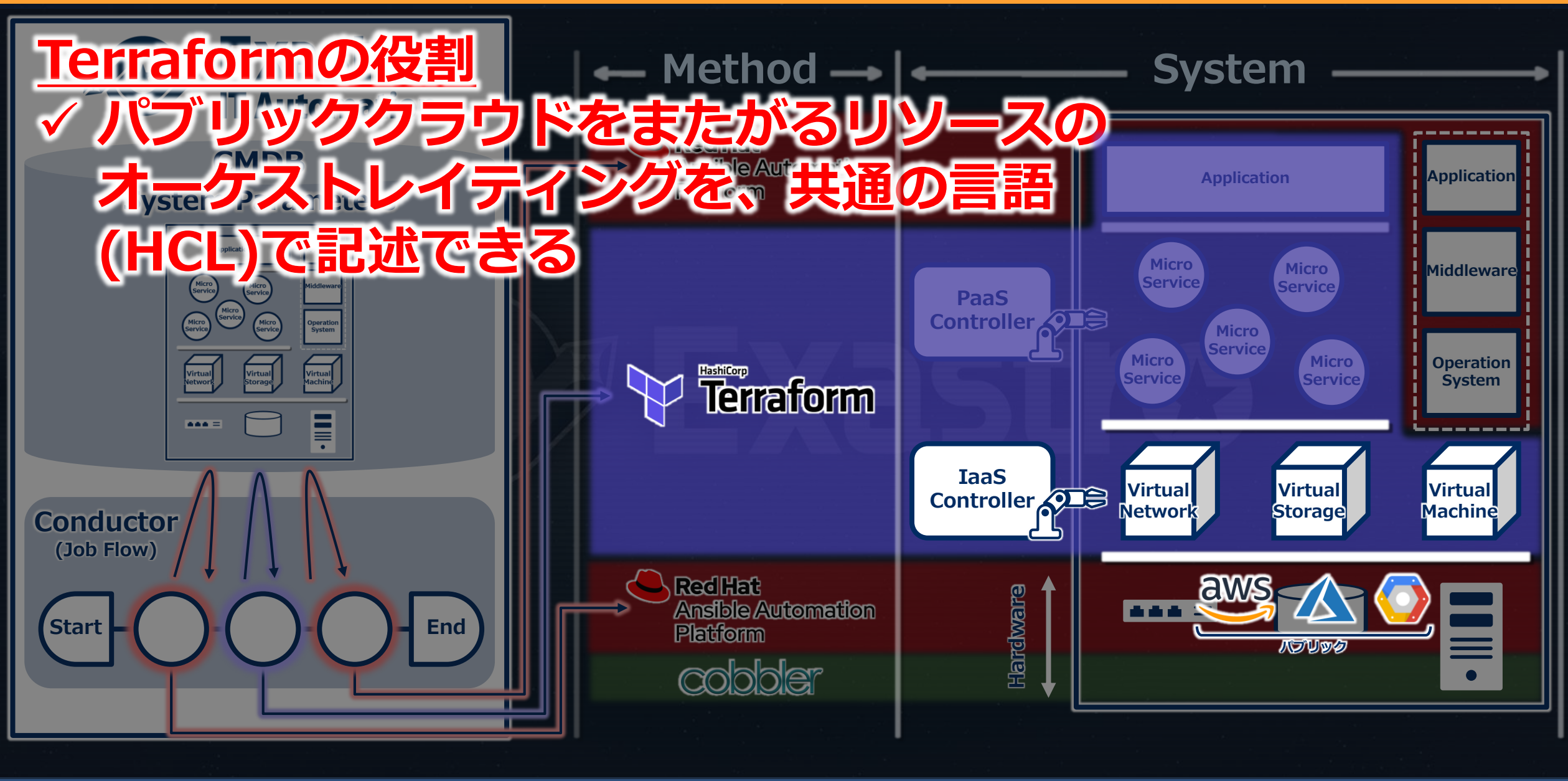
Exastro IT Automation：システム情報をデジタル管理するためのフレームワーク

構成管理ツールとして最も密に連携してきた「Red Hat Ansible Automation Platform」に加え、Ansibleと相互補完し合えるオーケストレーションツールとして「Terraform Enterprise」との連携活用が可能となっています

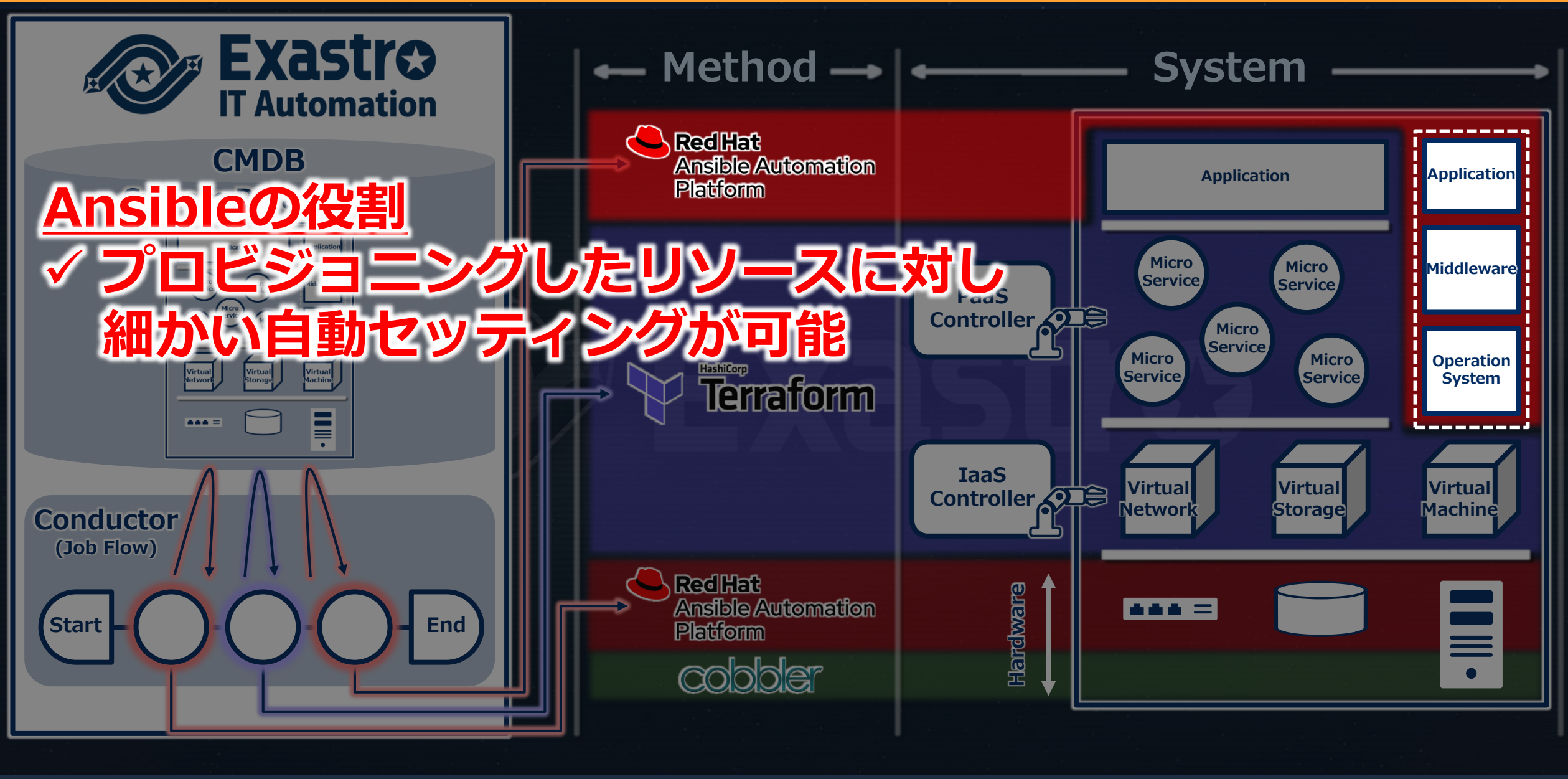
Exastro IT Automation：システムスタックと使用するメソッド



Exastro ITAからAnsibleとTerraformを相互補完して活用するということは？



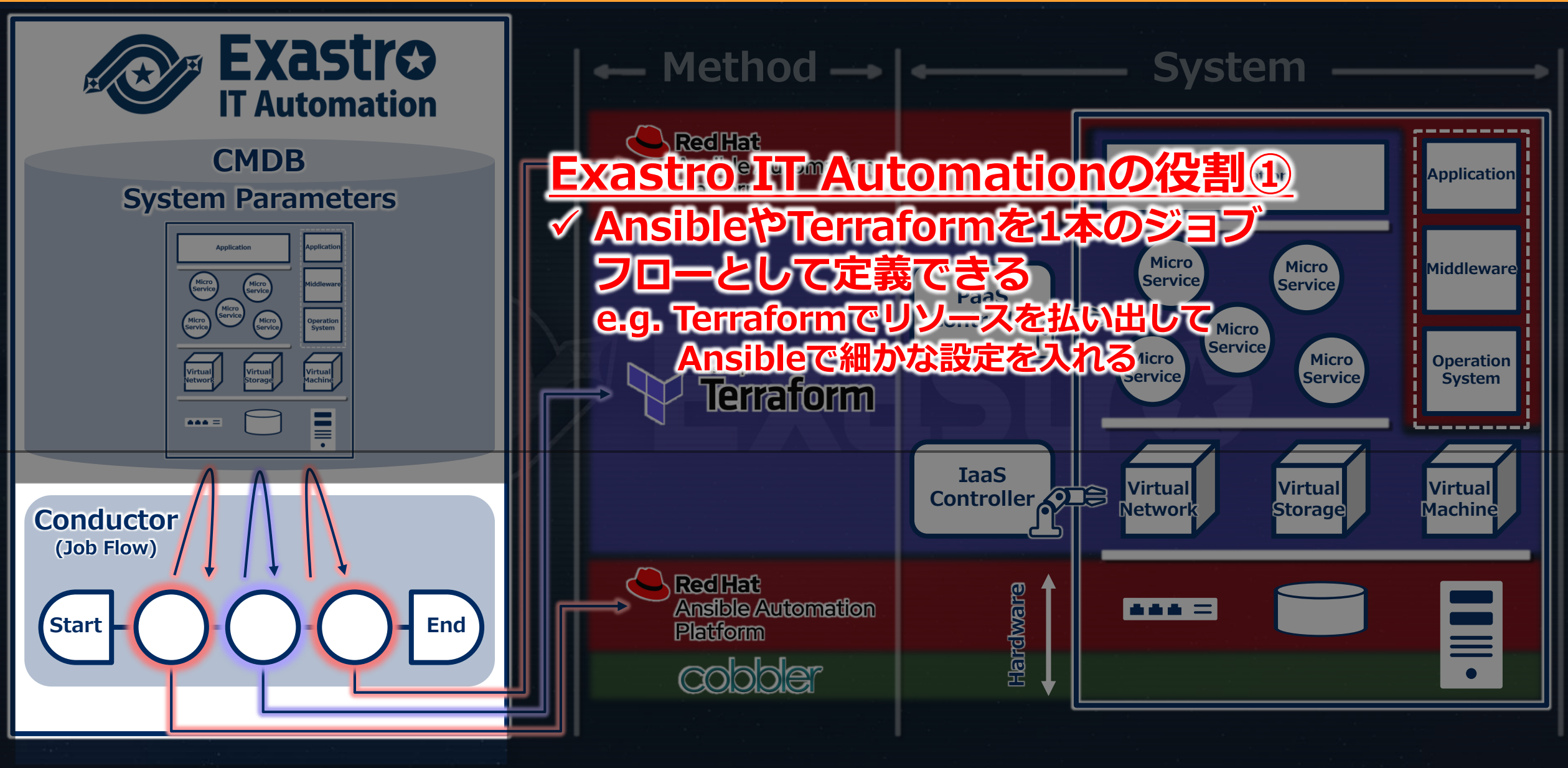
Exastro ITAからAnsibleとTerraformを相互補完して活用するということは？



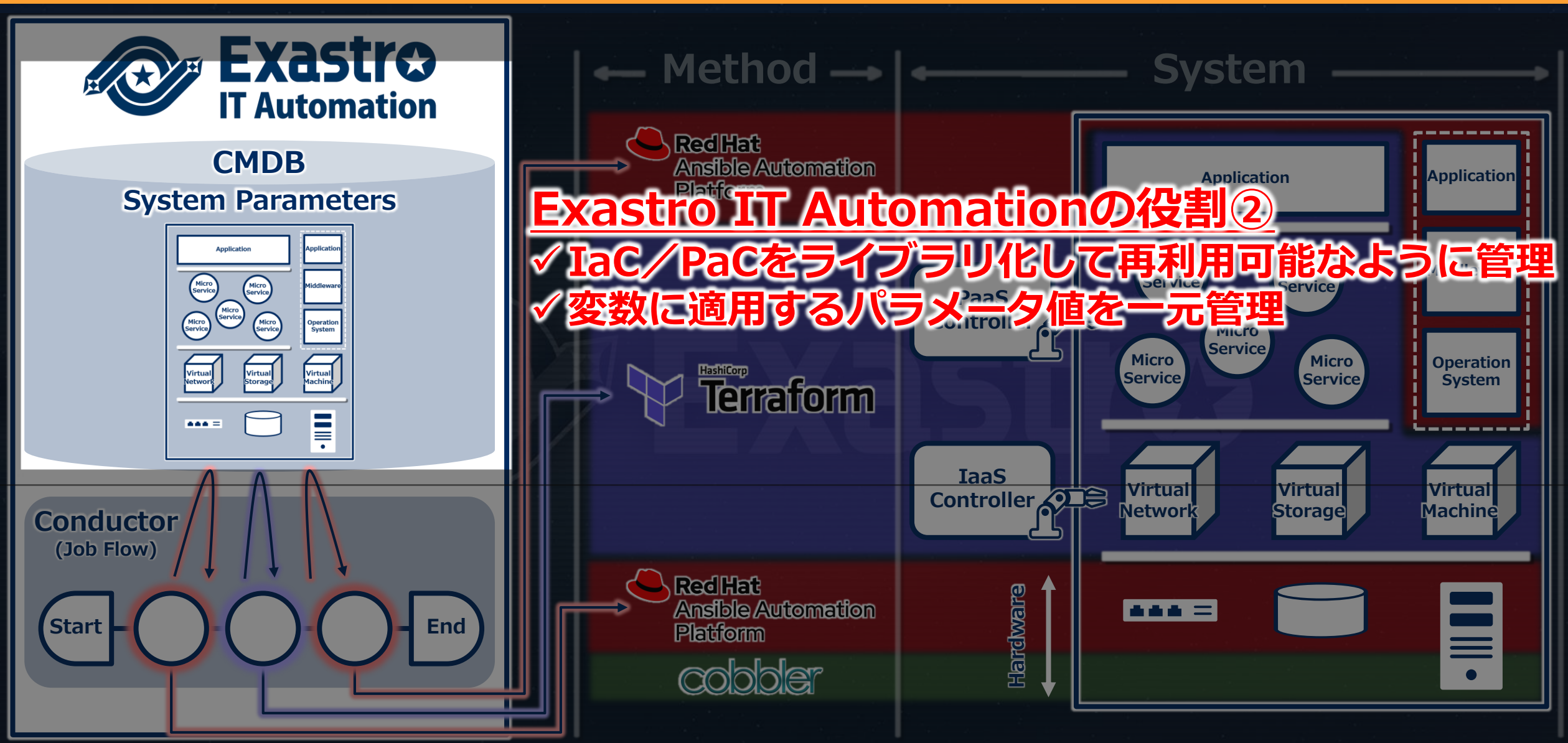
Ansibleの役割

✓プロビジョニングしたリソースに対し
細かい自動セッティングが可能

Exastro ITAからAnsibleとTerraformを相互補完して活用するということは？



Exastro ITAからAnsibleとTerraformを相互補完して活用するということは？



Exastro IT Automationの役割②

✓IaC／PaCをライブラリ化して再利用可能なように管理

✓変数に適用するパラメータ値を一元管理

[参考]Exastro IT Automation 関連記事

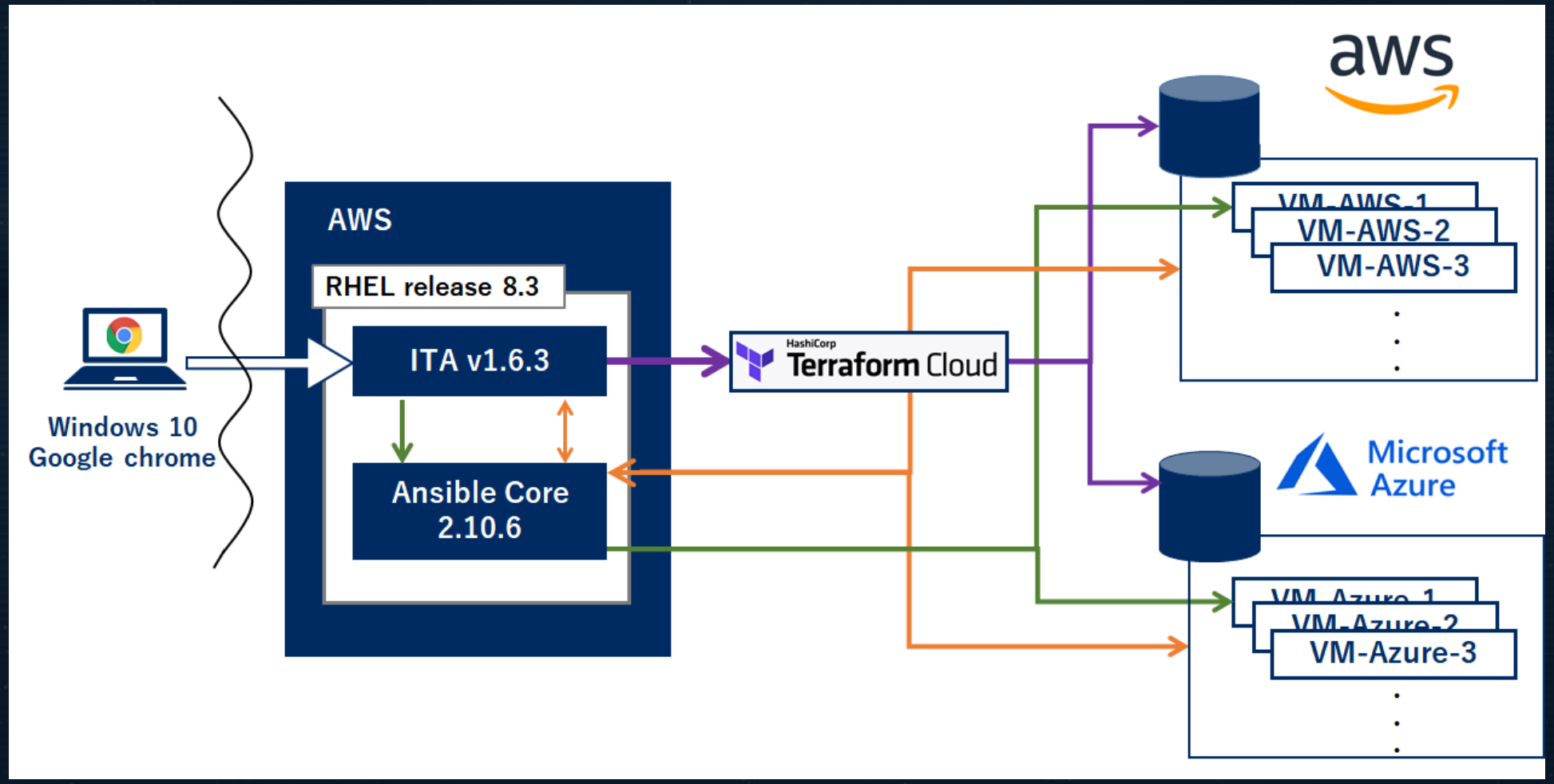
- Exastro IT Automationをインストールしてみた（v1.6.0）
- Exastro IT Automationを実際に動かしてみた(クイックスタート)
- Exastro IT Automation ver1.6.0のキホンの"キ“
- Exastro × Ansibleでやってみよう♪ ネットワーク機器の自動設定
- Exastroコミュニティ (Github)

デモ

デモ環境の構成イメージ

詳細は以下Qiita記事へ。

「マルチクラウドプロビジョニング(AWS＋Azure)を「Exastro×Terraform×Ansible」で実施してみた」
https://qiita.com/standsetx/items/5e20e9c30bb5efa9b12f



初期状態について

～デモ～

デモ実施前の状態

✓AWS：何も起動していない状態

✓Azure：何も起動していない状態

まずは実行してみよう

～デモ～

設定済みのConductorを実行してみる

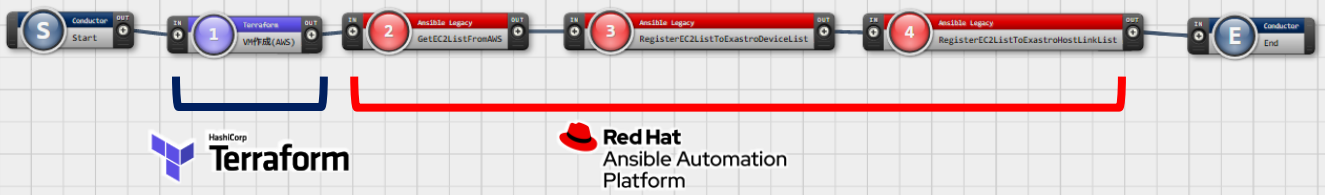
✓AWS：3インスタンスデプロイ→Webサーバセッティング

✓Azure：3インスタンスデプロイ→Webサーバセッティング

Conductor(ジョブフロー)について

✓3つのサブConductorから構成されるConductor

サブConductor①：AWSプロビ＋機器一覧登録



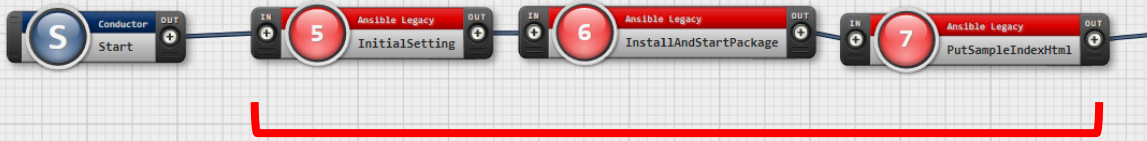
HashiCorp Terraform

Red Hat Ansible Automation Platform

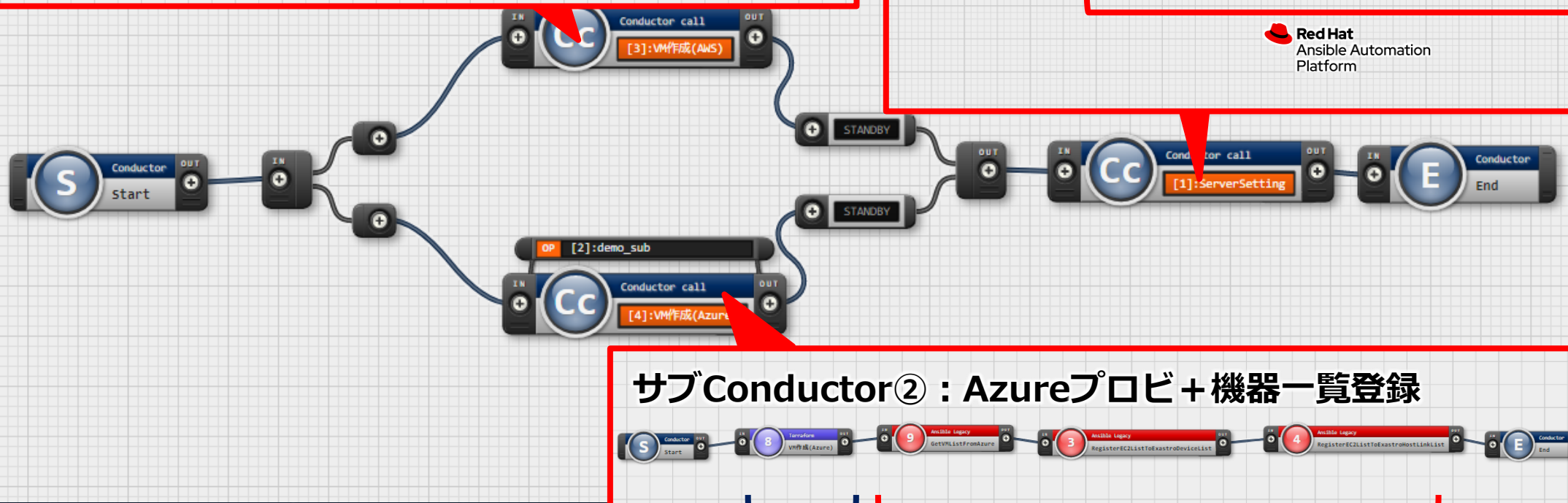
サブConductor③：

・全インスタンス(AWS+Azure)をWebサーバ化

・コンテンツの配布



Red Hat Ansible Automation Platform



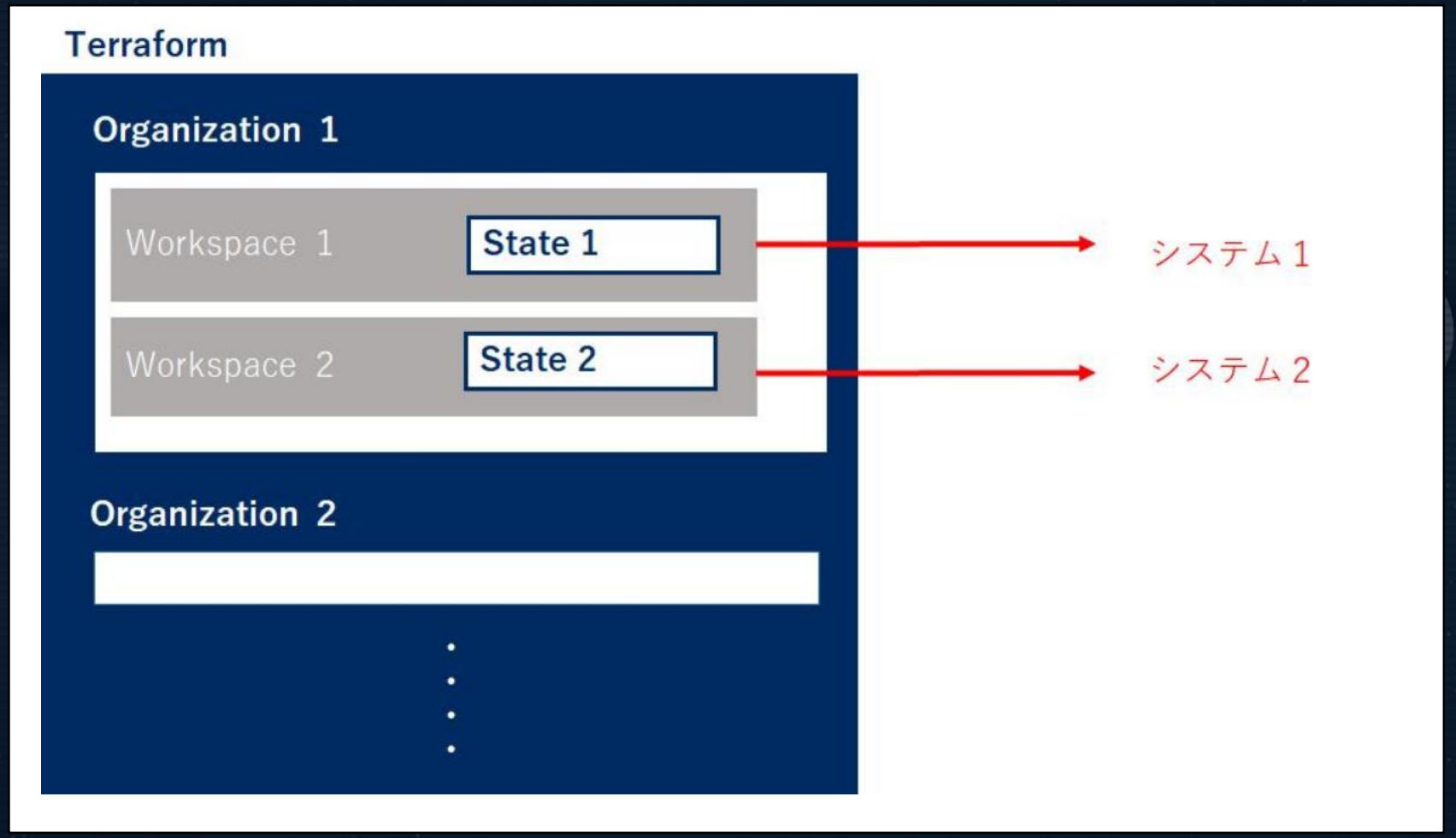
サブConductor②：Azureプロビ＋機器一覧登録

HashiCorp Terraform

Red Hat Ansible Automation Platform

Terraformについて

✓リソースのデプロイを行うために“tfstate”と呼ばれるファイルを作成
✓stateファイルはOrganization及びWorkSpaceに所属する



Terraform Driverの利用

～デモ～

Terraformドライバを利用して以下のセッティング

- ✓接続情報（Token)
- ✓Organizationの登録
- ✓WorkSpaceの作成
- ✓Movementの設定

Terraform Module素材(IaC)について

AWS用：

✓aws_create_instance_variables.tf ・・・変数定義ファイル

✓aws_create_instance.tf ・・・AWSインスタンス作成用リソース定義ファイル

Azure用：

✓azure_create_instance_variables.tf ・・・変数定義ファイル

✓azure_create_instance.tf ・・・Azureインスタンス作成用リソース定義ファイル

（参考）aws_create_instance_variables.tf（変数定義用）

aws_create_instance_variables.tf

```
variable "access_key" {}
variable "secret_key" {}
variable "region" {}
variable "ami" {}
variable "key_name" {}
variable "security_group" {}
variable "tags_name" {}
variable "hello_tf_instance_count" {
    default = 2
}
variable "hello_tf_instance_type" {
    default = "t2.micro"
}
```

（参考） aws_create_instance.tf（リソース定義ファイル）

aws_create_instance.tf

```
terraform {
  required_version = "~> 0.12"
}

provider "aws" {
  access_key = var.access_key
  secret_key = var.secret_key
  region = var.region
}

resource "aws_instance" "hello-tf-instance" {
  ami             = var.ami
  key_name        = var.key_name
  security_groups = [var.security_group]
  tags = {
    Name = "${var.tags_name}-${count.index+1}"
  }
  count = var.hello_tf_instance_count
  instance_type = var.hello_tf_instance_type
}
```

（参考） azure_create_instance_variables.tf （変数定義用）

azure_create_instance_variables.tf

```
variable "subscription_id" {}
variable "tenant_id" {}
variable "client_id" {}
variable "client_secret" {}
variable "resource_group_name" {}
variable "security_group" {}
variable "location" {}
variable "Vnet_name" {}
variable "Vnet_address_space" {}
variable "subnet_name" {}
variable "address_prefixes" {}
variable "public_ip_name" {}
variable "allocation_method" {}
variable "domain_name_label" {}
variable "network_interface_name" {}
variable "NIC_name" {}
variable "VM_name" {}
variable "VM_size" {}
variable "publisher" {}
variable "offer" {}
variable "sku" {}
variable "source_image_version" {}
variable "admin_username" {}
variable "ssh_public_key" {}
variable "os_disk_name" {}
variable "caching" {}
variable "storage_account_type" {}
```

（参考） azure_create_instance.tf（リソース定義ファイル）

azure_create_instance.tf

```
provider "azurerm" {
  features {}
  subscription_id = var.subscription_id
  client_id       = var.client_id
  client_secret   = var.client_secret
  tenant_id       = var.tenant_id
}

resource "azurerm_resource_group" "hogehoge" {
  name = var.resource_group_name
  location = var.location
}

resource "azurerm_network_security_group" "hogehoge" {
    name =var.security_group
    location = azurerm_resource_group.hogehoge.location
    resource_group_name = azurerm_resource_group.hogehoge.name

    security_rule {
        name                       = "SSH"
        priority                   = 1001
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "22"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }
    security_rule {
```

Terraform Driverの利用②

～デモ～

Terraformドライバを利用して以下のセッティング

✓パラメータシート

✓代入値自動登録設定

[Tips] パラメータシートについて

✓Terraform用パラメータシートとAnsible用パラメータシートは別で作成

HashiCorp Terraform

「パラメータシート（**オペレーションあり**）」で作成する

Red Hat Ansible Automation Platform

「パラメータシート（**ホスト/オペレーションあり**）」で作成する

Conductorの利用②

～デモ～

パラメータシートを変更して再デプロイしてみる

✓AWS：5インスタンスデプロイ→Webサーバセッティング

✓Azure：1インスタンスデプロイ→Webサーバセッティング

Ansible Driverの利用

～デモ～

Ansibleドライバを利用して以下のセッティング

✓Movement

✓パラメータシート

✓代入値自動登録

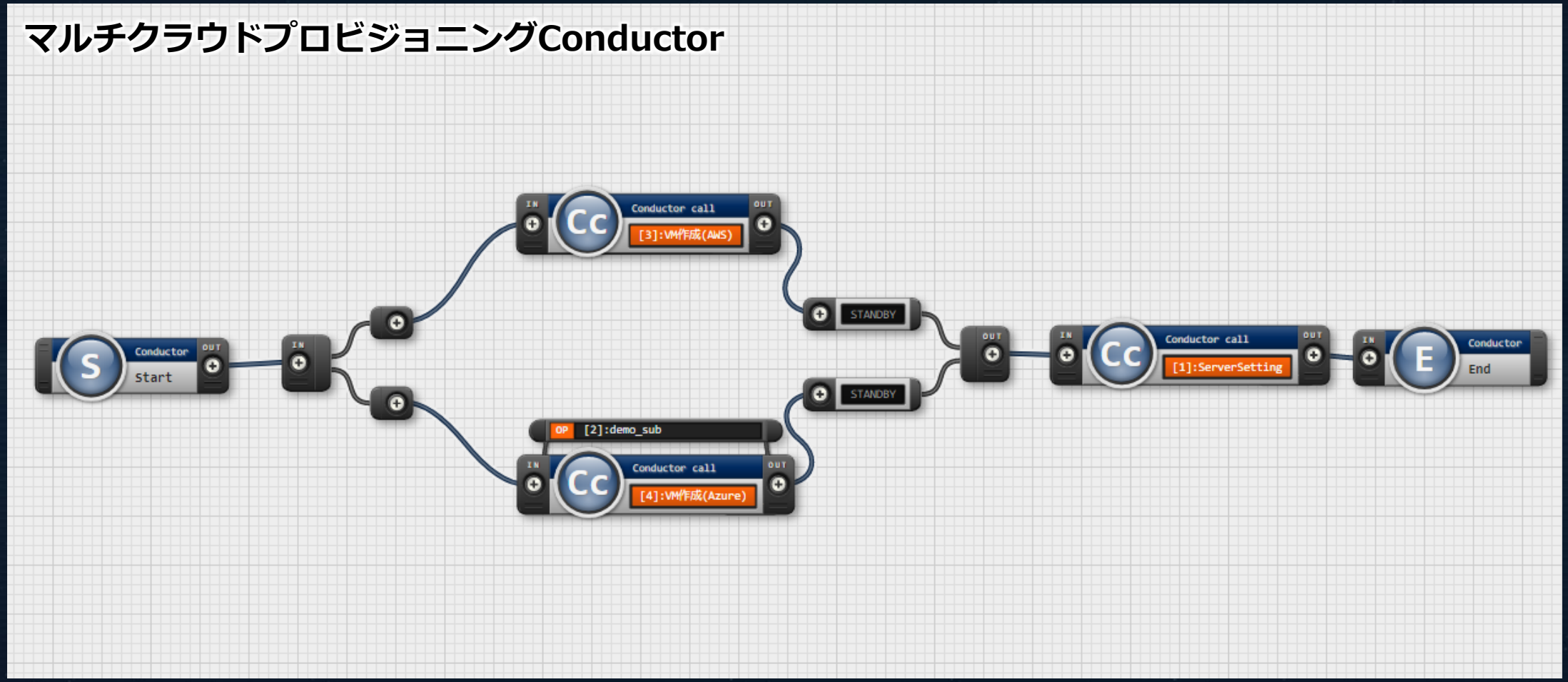
Conductorの設定

～デモ～

Conductorの準備
✓サブConductor/Movement
✓メインConductor

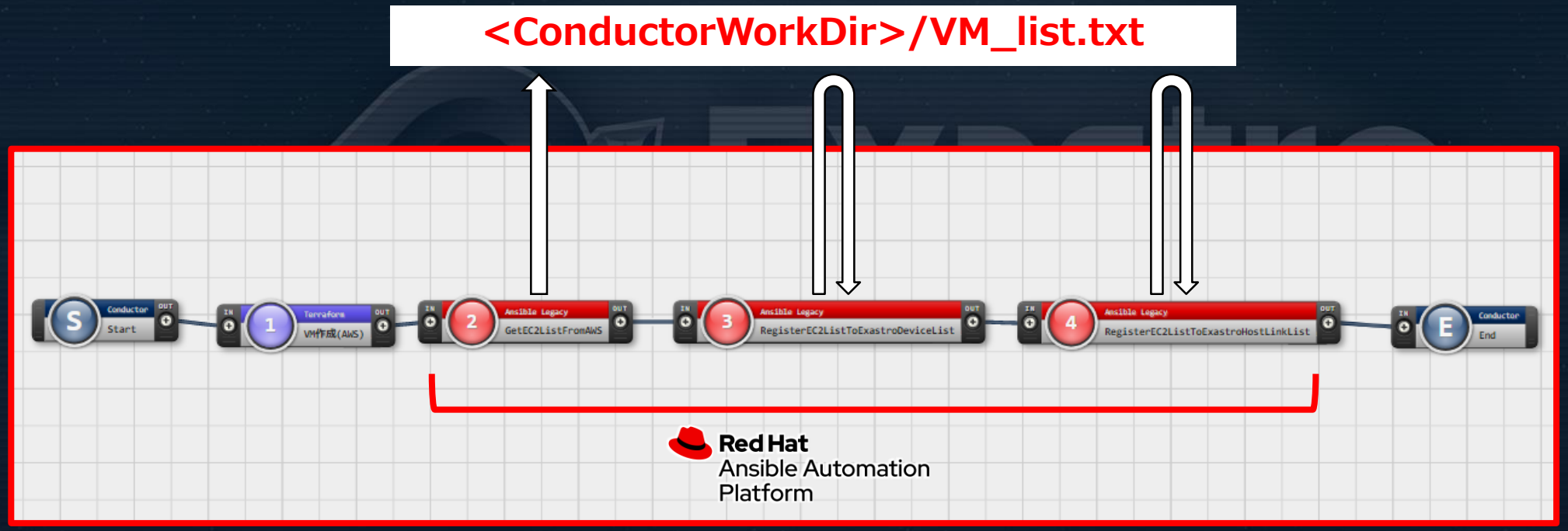
（再掲）Conductor(ジョブフロー)について

✓3つのサブConductorから構成されるConductor



[Tips] Conductor内でデータを受け渡しする

✓Movement間で情報/ファイルの受け渡し場所に“__conductor_workflowdir__”が利用可能
✓Conductor毎に払い出される




Exastro