REFERENCE ARCHITECTURE

Enterprise Data Center Infrastructure

February 2019

Deploying IBM Cloud Private* on VMware vSAN ReadyNodes*

Authors

Małgorzata Rembas
Cloud Solutions Architect, DCG, DSG, SDSG, Intel

Dominika Krzyszczyk
Cloud Solutions Engineer, DCG, DSG, SDSG, Intel

Karol Szwaj
Cloud Solutions Engineer, DCG, DSG, SDSG, Intel

Filip Skirtun
Cloud Solutions Engineer, DCG, DSG, SDSG, Intel

Executive Summary

IBM Cloud Private is a scalable cloud platform that runs on enterprise infrastructure and facilitates multi-cloud deployments. It's built on open source frameworks, like Docker* and Kubernetes*, and it provides the following foundational services:

- Hybrid cloud–based compute (rapid deployment, operating expense [OpEx] benefits, elasticity, scalability, and multi-cloud management and orchestration capabilities)
- Microservices architecture and integrated DevOps tools for rapid application modernization
- Containerization for easy, consistent deployment of any workload
- Security, logging, and monitoring infrastructure for easier administration

Container-based microservices architecture has become increasingly attractive to modern businesses that want to digitally transform their data centers with a modern application and technology infrastructure. Containers provide a lightweight complement to virtual machines (VMs) that are portable, run anywhere, are fast and easy to deploy, and can support both traditional apps and microservices. Containers are also attractive from a business perspective because they can be used with VMs to reduce hardware, licensing, and maintenance costs, accelerate time to market, increase productivity, and enable hybrid clouds.

The Intel® Reference Solution for IBM Cloud Private* enables IT organizations like yours to deploy a container solution cost effectively, quickly, and without adding complexity. The combination of IBM Cloud Private and VMware vSAN* on Intel® hardware can provide an economical, enterprise-grade way forward for digital transformation. In addition, the Intel Reference Solution for IBM Cloud Private enables you to quickly deploy IBM Cloud Private on VMware vSAN on a performance-optimized infrastructure built on Intel® Xeon® Scalable processors, Intel® 3D NAND Solid State Drives (SSDs), Intel® Optane™ DC SSDs, and the Intel® Ethernet 700 Series.

This reference architecture (RA) will show you how to prepare, provision, deploy, and manage an IBM Cloud Private solution. The intended audience for this RA includes system administrators and system architects. Some experience with Docker and Kubernetes is recommended.

Table of Contents

IBM, Intel, VMware, Docker*, and Kubernetes*

Beyond IBM Cloud Private software and Intel hardware, the Intel Reference Solution for IBM Cloud Private harnesses the power of VMware vSAN ReadyNodes*, Docker containers, and Kubernetes container orchestration to provide a comprehensive, reliable cloud solution.

IBM Cloud Private

IBM Cloud Private is a scalable cloud platform that is built on a container architecture based on Kubernetes. IBM Cloud Private includes management services like logging, monitoring, access control, and event management, and it brings cloud-native container capabilities to enterprise IT for all container use cases. Enterprises use the platform for three main use cases:

- Developing and running production cloud-native applications in a private cloud
- More securely integrating and using data and services from sources external to the private cloud
- Refactoring and modernizing heritage enterprise applications

Intel Hardware

Intel and IBM chose Intel Xeon Scalable processors for the Intel Reference solution for IBM Cloud Private because these processors support the most demanding workloads. Beyond compute based on Intel Xeon processors, the Intel Reference solution for IBM Cloud Private uses Intel SSDs and Intel Optane DC SSDs for performant all-flash storage and Intel® Ethernet Network Adapters for low-latency performance across the storage, management, and virtualization networks that undergird the solution.

VMware vSAN ReadyNodes

The Intel Reference Solution for IBM Cloud Private* uses VMware vSAN ReadyNodes to deliver hyper-converged infrastructure and serve as the foundation for a transformed, software-defined data center. Verified Intel solutions—such as the Intel Reference Solution for IBM Cloud Private—are certified for VMware vSAN ReadyNode* and are tightly specified by Intel and VMware to deliver balanced and optimized performance.

Docker Containers

Docker containers provide lightweight, standalone, executable software packages with everything needed to run an application, regardless of the infrastructure or operating system.

Kubernetes

IBM Cloud Private provides an open container platform based on Kubernetes for orchestration. The platform enables automated deployment, scaling, and management of containerized applications.

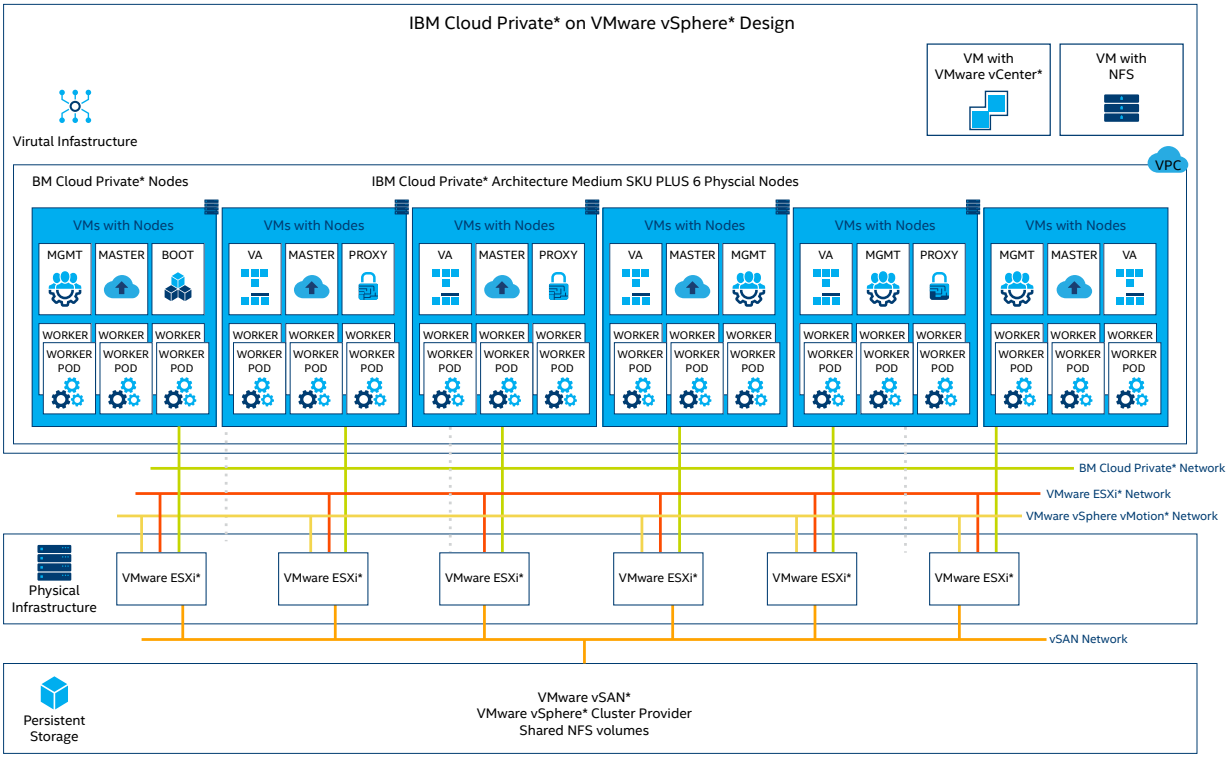


Figure 1. IBM Cloud Private* system architecture for a medium-sized deployment

Intel Solution for IBM Cloud Private System Architecture

Figure 1 shows the IBM Cloud Private system architecture with nodes in VMs on VMware ESXi* servers based on Intel hardware.

IBM Cloud Private Cluster Overview

An IBM Cloud Private cluster has four main classes of nodes: boot, master, worker, and proxy, with two optional nodes—management and vulnerability advisor.

Boot Node

A boot, or bootstrap, node is used for running installation, configuration, node scaling, and cluster updates. Only one boot node is required for any cluster. You can use a single node for both master and boot.

Master Node

A master node provides management services and controls the worker nodes in a cluster. Master nodes host processes that are responsible for resource allocation, state maintenance, scheduling, and monitoring. Multiple master nodes can be configured in a high availability (HA) environment to allow for failover if the leading master host fails. Hosts that can act as the master are called master candidates.

Worker Node

A worker node is a node that provides a containerized environment for running tasks. As demands increase, more worker nodes can easily be added to your cluster to improve performance and efficiency (see the IBM Cloud Private cluster sizing guidelines in Table 2). A cluster can contain any number of worker nodes, but a minimum of one worker node is required.

Proxy Node

A proxy node is a node that transmits external requests to the services created inside your cluster. Multiple proxy nodes can also be deployed in an HA environment to allow for failover if the leading proxy host fails. While you can use a single node as both master and proxy, it is best to use dedicated proxy nodes to reduce the load on the master node. A cluster must contain at least one proxy node if load balancing is required inside the cluster.

Management Node

A management node is an optional node that only hosts management services such as monitoring, metering, and logging. By configuring dedicated management nodes, you can prevent the master node from becoming overloaded. You can enable a management node only during IBM Cloud Private installation.

Vulnerability Advisor Node

A vulnerability advisor (VA) node is an optional node that is used for running the VA services, which can be resource-intensive. If you use the VA service, specify a dedicated VA node.

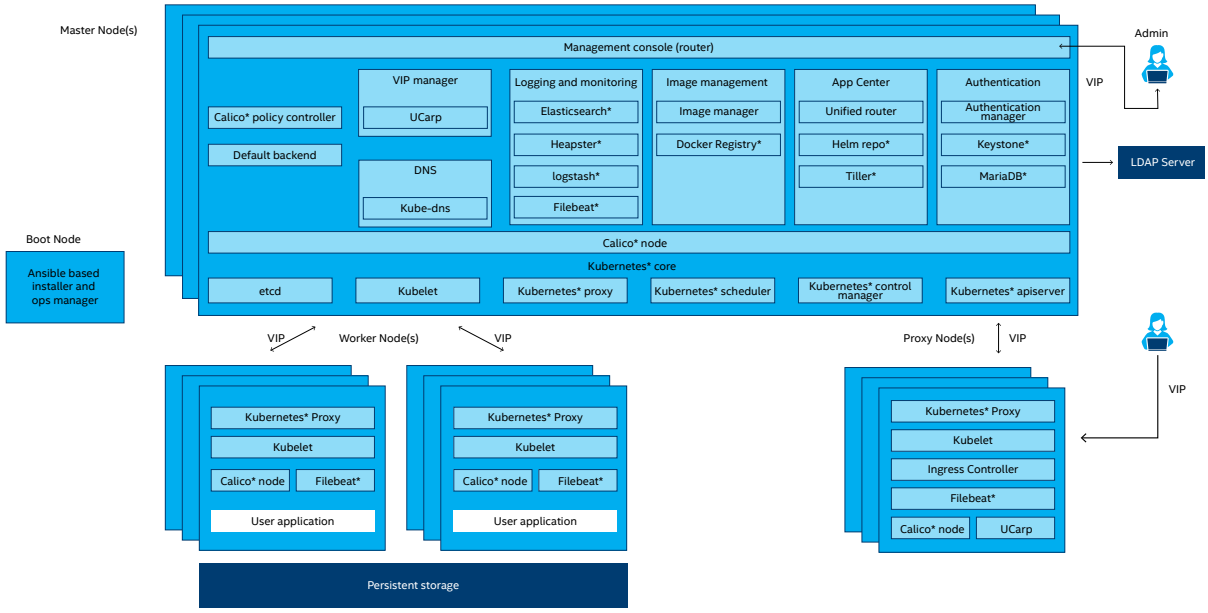


Figure 2. Interrelation of IBM Cloud Private* nodes

IBM Cloud Private Components

IBM Cloud Private provides a container runtime (Docker) and a container orchestration platform (Kubernetes) along with an integrated layer of additional services.

Other components of an IBM Cloud Private cluster work alongside these main components to provide services such as authentication, storage, networking, logging, and monitoring. A cluster-management console is also provided, which allows for centralized management of these services.

Software Components

Principal software components for an Intel Reference Solution for IBM Cloud Private deployment are:

- Red Hat* Enterprise Linux* (RHEL*) v7.5
- IBM Cloud Private version 3.1.1
- VMware vSphere ESXi 6.7 U1*
- VMware vCenter Server Appliance 6.7 U1*
- VMware vSAN ESXi 6.7 U1*

Table 1. Intel® Reference Solution for IBM Cloud Private* component software and versions

SOFTWARE NAME	VERSION
VMware vSphere ESXi*	ESXi 6.7 U1
VMware vCenter*	VMware vCenter Server Appliance 6.7 U1*
VMware vSAN*	ESXi 6.7 U1
IBM Cloud Private* Enterprise Edition	3.1.1
IBM Cloud Private Master Node	Red Hat* Enterprise Linux* (RHEL*) v7.5
IBM Cloud Private Proxy Node	RHEL v7.5
IBM Cloud Private Worker Node	RHEL v7.5
IBM Cloud Private Management Node	RHEL v7.5
IBM Cloud Private VA Node	RHEL v7.5
IBM Cloud Private Boot Node	RHEL v7.5

Hardware Components and Configuration

The SKU Size Calculator (Table 2) enables determining how many physical nodes in which hardware configuration (Dev/Test, Base, or Plus) are required to support a given number of VMs with workloads on the small, medium, or large IBM Cloud Private clusters. The minimum and maximum number of workers should be considered when deciding how many physical nodes to deploy.

Use the following section to help determine the ideal cluster size and configuration for your environment.

Cluster Sizing

Table 2. Number of physical nodes required to support different ranges of VM-based workloads by cluster size and configuration for the Intel® Reference Solution for IBM Cloud Private*

	SMALL IBM CLOUD PRIVATE*		MEDIUM IBM CLOUD PRIVATE		LARGE IBM CLOUD PRIVATE
	Max. 20 Workers	Min. 5 Workers	Max. 70 Workers	Min. 7 Workers	Max. 150 Workers
DEV/TEST CONFIGURATION	4	5	17	–	–
BASE CONFIGURATION	4	4	9	4	17
PLUS CONFIGURATION	4	4	6	4	12

Intel Reference Solution for IBM Cloud Private Configurations

Use the following tables to help determine the ideal Intel Reference Solution for IBM Cloud Private configuration for your environment.

Dev/Test Configuration

Table 3. Intel® Reference Solution for IBM Cloud Private* Dev/Test configuration component hardware

ROLE	PLATFORM	CONFIGURATION
VMware vSphere ESXi*	Intel® Server Board S2600WF0	2 x Intel® Xeon® Silver 4114 processor (2.20 GHz, 10 cores, 20 threads) or a higher number Intel Xeon Scalable processor 12 x 16 GB 2,666Hz DDR4—192 GB RAM 2 x 480 GB Intel® SSD DC S3520 (M.2, SATA 3.0, 6 gigabit per second [Gb/s]) 1 x 375 GB Intel® Optane™ SSD DC P4800X (2.5-in. PCIe*) 3 x 2 TB Intel SSD DC P4510 (2.5-in. PCIe, NVM Express* [NVMe*] 3.0 x4) 1 x Intel® Ethernet Converged Network Adapter X710-DA2 1 x network interface controller (NIC)-integrated 1 gigabit Ethernet (GbE)

Base Configuration

Table 4. Intel® Reference Solution for IBM Cloud Private* Base configuration component hardware

ROLE	PLATFORM	CONFIGURATION
VMware vSphere ESXi*	Intel® Server Board S2600WF0	2 x Intel® Xeon® Gold 5120 processor (2.20GHz, 14 cores, 28 threads) or a higher number Intel Xeon Scalable processor 12 x 32 GB 2,666MHz DDR4—384 GB RAM 2 x 480 GB Intel® SSD DC S3520 (M.2, SATA 3.0, 6 Gb/s) 2 x 375 GB Intel® Optane™ SSD DC P4800X (2.5-in. PCIe*) 6 x 2 TB Intel SSD DC P4510 (2.5-in. PCIe, NVMe* 3.0 x4) 2 x Intel® PCIe Extender AXXP3SWX08080 8-Port PCIe Gen3 x8 Switch AIC 1 x Intel® Ethernet Converged Network Adapter X710-DA2 1 x NIC-integrated 1 GbE

Plus Configuration

Table 5. Intel® Reference Solution for IBM Cloud Private* Plus configuration component hardware

ROLE	PLATFORM	CONFIGURATION
VMware vSphere ESXi*	Intel® Server Board S2600WF0	2 x Intel® Xeon® Gold 6148 processor (2.50 GHz, 20 cores, 40 threads) or a higher number Intel Xeon Scalable processor 24 x 32 GB 2,666MHz DDR4—768 GB RAM** 2 x 480 GB Intel® SSD DC S3520 (M.2, SATA 3.0, 6Gb/s) 2 x 375 GB Intel® Optane™ SSD DC P4800X (2.5-in. PCIe*) 8 x 2 TB Intel SSD DC P4510 (2.5-in. PCIe, NVMe* 3.0 x4) 2 x Intel® PCIe Extender AXXP3SWX08080 8-Port PCIe Gen3 x8 Switch AIC 1 x Intel® Ethernet Converged Network Adapter X710-DA4 Or 2 x 25 GbE Intel® Ethernet Controller XXV710 1 x NIC-integrated 1 GbE

** A cost-optimized configuration can use 512 GB RAM (2 x CPU, 16 x 32 GB 2,666 MHz DDR4). However, this is an unbalanced configuration will decrease memory-access performance.

BIOS Settings

BIOS version: SE5C620.86B.00.01.0013.030920180427, which includes:

- System BIOS: 00.01.0013
- Intel® Management Engine (Intel® ME) firmware: 04.00.04.294
- Baseboard management controller (BMC) firmware: 1.43.91f76955
- FRUSDR*: 1.43

BIOS version R00.01.R0012 microcode 0x43 and 0x13f update addressed (Intel-SA-00088) Meltdown and Spectre vulnerabilities in the processor.

Table 6. VMware ESXi* node BIOS settings

TECHNOLOGIES	SETTING
Trusted Platform Module (TPM) 2.0	Enabled
Intel® Trusted Execution Technology (Intel® TXT)	Enabled
Intel® Hyper-Threading Technology (Intel® HT Technology)	Enabled
Intel® Turbo Boost Technology	Enabled
Intel® Speed Shift Technology—hardware P-states (HWP) native	Enabled
Intel® Volume Management Device (Intel® VMD)	Disabled
Intel® Virtualization Technology for Directed I/O (Intel® VT-d), virtualization	Enabled
Power-management settings	Performance, workload input/output (I/O) intensive
PCIe*, Single-Root I/O Virtualization (SR-IOV) support	Enabled
Integrated I/O	Intel VT-d
Advanced	Set fan profiles to performance PCIe configuration, NIC configuration: disable 3, 4

Networking Components

Table 7. Network-infrastructure components

ROLE	QTY.	PLATFORM	DESCRIPTION
High performance top-of-rack (ToR) switches	2	Arista DCS-7050SX2-72Q 40 GbE SFP+*	arista.com/en/products/7050x-series
BMC switch	1	Arista DCS-7010T 48 ports 1 GbE*	arista.com/en/products/7010-series

Intel Hardware Details

Intel processing, networking, and storage hardware provide the performance-optimized foundation for the Intel Reference Solution for IBM Cloud Private.

Intel® Xeon® Scalable Processors

Intel Xeon Scalable processors provide a new foundation for more secure, agile, multi-cloud data centers. This platform provides businesses with breakthrough performance to handle system demands ranging from entry-level cloud servers to compute-hungry tasks including real-time analytics, virtualized infrastructure, and high-performance computing (HPC). This processor family includes technologies for accelerating and securing specific workloads:

- **Intel® Advanced Vector Extensions 512 (Intel® AVX-512)** expands the number of registers and adds instructions to streamline processing. The result is high performance of mixed workloads.
- **Intel® Transactional Synchronization Extensions New Instructions (Intel® TSX-NI)** provides high performance of multi-threaded workloads.
- **Intel® Trusted Execution Technology (Intel® TXT)** provides the necessary underpinnings to evaluate the computing platform and its security.
- **Intel® Platform Trust Technology (Intel® PTT)** provides more secure encryption keys in a Trusted Platform Module (TPM) integrated directly into the chipset.

Intel Xeon Scalable processors were selected for the RA because they meet the performance needs for demanding, mission-critical workloads.

Intel® Ethernet Network Adapter X710

Intel Ethernet Network Adapter X710 delivers excellent performance with a throughput of 40 Gb/s in a PCI Express* (PCIe*) v3.0 x8 slot.[1]

Optimized performance vectors and key uses include:

- **Small Packet Performance**: Achieves excellent throughput on smaller payload sizes
- **Virtualized Performance**: Alleviates hypervisor input/output (I/O) bottlenecks by providing flow separation for VMs
- **Network Virtualization**: Offloads network-virtualization overlays, including Virtual Extensible LAN (VXLAN), Network Virtualization Using Generic Routing Encapsulation (NVGRE), GENEVE*, Multiprotocol Label Switching (MPLS), and VXLAN GPE with Network Service Header (NSH)

Intel® Data Center SSDs

The Intel Reference Solution for IBM Cloud Private uses all-flash storage to minimize storage latency and maximize storage availability.

- **Intel SSD DC S3520**: Provides data integrity, performance consistency, and drive reliability for read-intensive workloads in the Intel Reference Solution for IBM Cloud Private, such as boot, web server, low-rate operational databases, and analytics. IBM and Intel selected the Intel SSD DC S3520 for the Intel Reference Solution for IBM Cloud Private because it provides an ideal combination of performance and cost.
- **Intel SSD DC P4510**: Is a cloud-inspired Intel 3D NAND SSD optimized for cloud infrastructures, and it offers high quality, reliability, advanced manageability, and serviceability to minimize service disruptions for the Intel Reference Solution for IBM Cloud Private.
- **Intel Optane SSD DC P4800X**: Allows bigger, more affordable datasets and helps eliminate data center storage bottlenecks for the Intel Reference Solution for IBM Cloud Private. IBM and Intel selected Intel Optane SSD DC P4800X for the Intel Reference Solution for IBM Cloud Private in order to accelerate application performance, reduce transaction costs for latency sensitive workloads, and help improve overall total cost of ownership (TCO).

Configuration Details

This section details some of the specific configuration steps taken in producing this reference architecture. Additional configuration details can be found in VMware and IBM documentation (linked to as appropriate below).

Configuring VMware vSphere*

Begin your Intel Reference Solution for IBM Cloud Private installation by deploying VMware ESXi and connecting it to the management network. Next, install VMware vCenter* using the VMware vCenter Server Appliance Installer Media.

1. Download the ISO image containing the VMware vCenter Server 6.7U1 Installer Media from the https://my.vmware.com/web/vmware/details?productId=742&rPId=24636&downloadGroup=VC67U1 portal.
2. Confirm that the md5sum is correct for your downloaded ISO file.
3. Mount the ISO file and run the installer files. There are two installer modes available: Disk Image Mounter (graphical user interface [GUI]) and terminal mode. For ease of use, Disk Image Mounter is recommended.

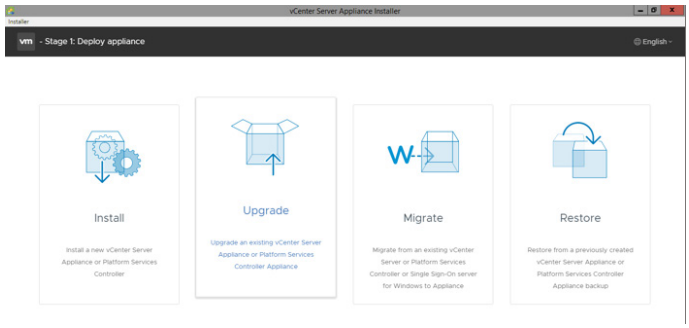


Figure 3. VMware vCenter Server Appliance* installer GUI

4. Select **Install**, and then complete the next nine steps. For this RA, we have selected the **Embedded Platform Services Controller** option and a **medium VSCA Deployment** size, and we've made use of the ability to bootstrap vSAN on a single host, placing vCenter Server Appliance on the newly created vSAN datastore.

After a successful installation of vCenter, follow these basic steps to complete its deployment:

1. Configure VMware vSphere Distributed Switch* networking.

   In vSphere, there are two methods of connecting VMs to physical networks. The first is based on vSphere standard switches; the second is based on vSphere Distributed Switch. This reference architecture uses vSphere Distributed Switch. This allows VMs to maintain their network settings even if they are migrated to a different host, as the settings of a distributed switch are propagated to all hosts associated with the switch.

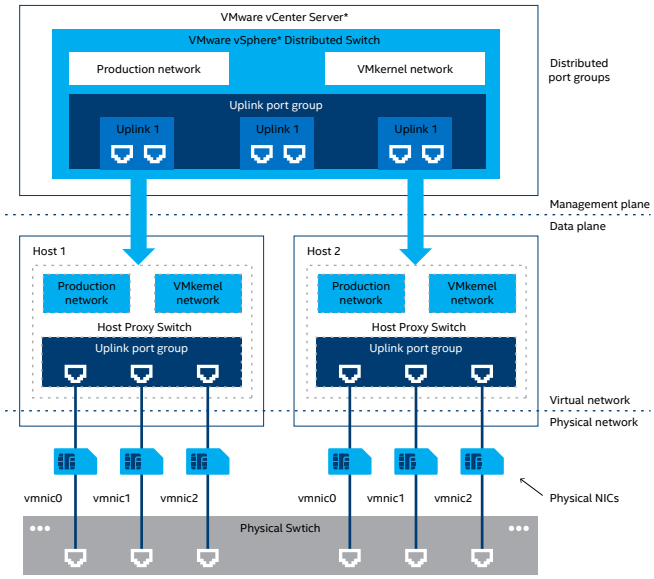


Figure 4. VMware vSphere* distributed switch architecture

vSphere Distributed Switch separates the data plane and the management plane. Management resides centrally on the vCenter Server and it lets users administer the networking configuration of all ESXi hosts in a data center.The data plane remains local to every host that is connected to the distributed switch as the host proxy switches. The networking configuration that you create on vCenter Server is automatically pushed down to all host proxy switches.

vSphere Distributed Switch has a lot of other features that distinguish it from a standard vSwitch. One example is the Network I/O Control feature, which allows users to provide quality of service (QoS) on the various traffic types, like the vSAN traffic.

vSphere Distributed Switch specifies two abstractions: Uplink Port Groups and Distributed Port Groups. The first one maps the physical NICs of each ESXi host to uplinks on the distributed switch; the second one provides network connectivity to VMs and forwards the VMkernel traffic. You can configure NIC teaming, failover, load balancing, VLAN, security, traffic shaping, and other policies on distributed port groups.

2. Add and configure VMkernel interfaces for VMware vSphere vMotion* and vSAN networks on each ESXi host.

   Each ESXi host needs to configure the VMkernel interfaces for vMotion and vSAN networks. Both of these networks are separated and reside on different VLANs. VMware vSphere vMotion provides the functionality of moving VMs automatically across the available pool of resources according to a set of rules, if defined by the administrator.

   In vSphere, you can easily create these rules on each ESXi host with the PowerCLI* tool, which is the PowerShell* interface for managing the whole VMware vSphere stack. Assuming you have created a vSphere Distributed Switch named “Dswitch,” then you can use the following commands:

```
New-VirtualPortGroup -Name “vSAN”
-VirtualSwitch “DSwitch” -VLanId 12

New-VMHostNetworkAdapter -VMHost
172.17.255.12 -PortGroup “vSAN”
-VirtualSwitch “DSwitch” -IP 192.168.8.12
-SubnetMask 255.255.255.0 -Mtu 9000
-VsanTrafficEnabled $true

New-VirtualPortGroup -Name “vMotion”
-VirtualSwitch “DSwitch” -VLanId 11

New-VMHostNetworkAdapter -VMHost
172.17.255.12 -PortGroup “vMotion”
-VirtualSwitch “DSwitch” -IP 192.168.4.12
-SubnetMask 255.255.255.0 -Mtu 9000
-VMotionEnabled $true
```

3. Configure a cluster with vSphere High Availability and VMware vSphere Distributed Resource Scheduler* for better HA and scheduling of VMs.

   A vSphere Distributed Resource Scheduler cluster is a collection of ESXi hosts and VMs with all shared resources and management interfaces. It offers three cluster-level resource-management capabilities: load balancing, power management, and VM placement. vCenter monitors distribution and usage of CPU and memory resources for all hosts and VMs in the cluster, and then vSphere Distributed Resource Schedulers compares these metrics with the current demand and the imbalance target for the desired resource utilization. vSphere Distributed Resource Scheduler then performs or recommends, depending how is it configured, VM migrations or new VM placement.

   vSphere High Availability monitors the pool of ESXi hosts and VMs, and, in the event of host failure, it migrates VMs to hosts with available capacity using vSphere vMotion. When you add new VMs to a vSphere High Availability cluster, vSphere High Availability checks if there are enough resources to power the VM on in case of host failure.

   This RA enables HA with host monitoring and vSphere Distributed Resource Schedulers with partially automated settings, where only the placement of VMs onto hosts is automated; the migration recommendations need to be manually applied.

4. Configure vSAN and claim disks for its cache and capacity tier.

VMware vSAN* Provisioning and Orchestration

In this setup, vSAN is configured in an all-flash mode, where one flash device is used for cache, while other flash devices are used for the capacity layer. This greatly improves the performance of VMs running on vSAN, in comparison to a hybrid vSAN configuration. In an all-flash configuration, the cache tier is used for the write cache.

Each ESXi host in the cluster has a configured VMkernel interface for vSAN traffic on separate distributed port groups. For this RA, we created vSAN disk groups by claiming each server’s disks; in our case, Intel Optane SSD DC P4800X disks are matched as cache tier and Intel SSD DC P4510 disks as capacity tier, and each disk group contains a maximum of one flash cache device and up to seven capacity disks.

Table 8 shows how vSAN is configured for each server in the cluster (SKU plus configuration); there are two disks groups with this configuration.

Table 8. VMware vSAN* configuration for this RA

DISK GROUP	STORAGE SKU	TIER	CAPACITY
Disk Group 1	1 x Intel® Optane™ SSD DC P4800X (2.5-in. PCIe*)	Cache tier	350 GB
	3 x Intel® SSD DC P4510 (2.5-in. PCIe 3.1)	Capacity tier	5.46 TB
Disk Group 2	1 x Intel Optane SSD DC P4800X (2.5-in. PCIe)	Cache tier	350 GB
	3 x Intel SSD DC P4510 (2.5-in. PCIe 3.1)	Capacity tier	5.46 TB

An all-flash configuration allows users to turn on the deduplication and compression modes, which can help improve the TCO by reducing the data stored on your physical disks.

Installing IBM Cloud Private

Follow these steps to deploy IBM Cloud Private. For detailed instructions, see the IBM documentation at ibm.com/support/knowledgecenter/SSBS6K_3.1.1/installing/install_containers.html.

1. Prepare a Red Hat Enterprise Linux (RHEL) 7.5 VM template.

   You need to prepare a RHEL 7.5, or another operating system, from the template available here, ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/supported_system_config/supported_os.html, which will be used as the base of all IBM Cloud Private nodes.

   In vCenter Server, create a new Linux VM named icp-temp—this name will be used later with the terraform deployment. The guest operating system version should be RHEL 7 (64-bit). During customization of the virtual hardware, you only need to specify virtual disk size, as the CPU and memory will be modified for each VM in later steps, so you can leave the default settings. For this RA's large environment, we have chosen 500 GB size for the template according to sizing provided by ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/installing/plan_capacity.html. In any case, use thin provisioning so the template does not use unnecessary disk space. This will also speed up creating new VMs from this template. The template VM also needs a single virtual network adapter, which should be connected to the network created for IBM Cloud Private VMs.

   Mount the RHEL installation image and start standard installation. Make sure to configure the Logical Volume Manager (LVM) disk sizes according to ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/supported_system_config/hardware_reqs.html. Configure networking accordingly for your setup; for this RA, we used a Dynamic Host Configuration Protocol (DHCP) server and set values in `/etc/sysconfig/network-scripts/ifcfg-ens192` to:

   ```
   BOOTPROTO=dhcp

   ONBOOT=yes
   ```

Now, you should configure access to RHEL yum repositories or Red Hat Satellite*, update RHEL to the latest patch level, and install the following tools and packages:

- Python 2.7.x*
- PyYAML*
- bind-utils
- nfs-utils
- Perl*
- open-vm-tools (VMware Tools* can be installed optionally via vCenter)
- ntp

IBM Cloud Private requires Docker to be installed on all cluster nodes. Per IBM Cloud Private documentation (ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/installing/install_docker.html), you can choose to either let the IBM Cloud Private installer set it up or install it manually from package provided with IBM Cloud Private—IBM Cloud Private 3.1.1 Docker for Linux (x86_64) from IBM Software Access Catalog Downloads (ibm.com/partnerworld/program/benefits/software-access-catalog).

We chose to pre-install Docker on the template, using the Docker install binary that comes with IBM Cloud Private. It is the version of Docker that has been tested with the given release of IBM Cloud Private and is supported by IBM—IBM Cloud Private 3.1.1 is matched with Docker 18.03. Download the package to the template VM and run these commands to install:

```
chmod +x icp-docker-18.03.1_x86_64.bin

sudo ./icp-docker-18.03.1_x86_64.bin
--install
```

For the pre-install Docker, it must be configured with devicemapper or overlay2 storage driver. The storage driver for the supplied IBM Cloud Private Docker package is set to loop-lvm by default. For production deployments, you must change to a different storage option: either direct-lvm or overlay2. For this RA, we have decided to use devicemapper in supported direct-lvm mode, so we have provided another disk to Docker. The recommended starting disk size is 100 GB.

Create LVM for Docker on a new disk and configure thin pools:

```
pvcreate /dev/sdb

vgcreate docker /dev/sdb

lvcreate --wipesignatures y -n thinpool
docker -l 95%VG

lvcreate --wipesignatures y -n
thinpoolmeta docker -l 1%VG

lvconvert -y --zero n -c 512K --thinpool
docker/thinpool --poolmetadata docker/
thinpoolmeta
```

Set up direct-lvm:

```
mkdir -p /etc/docker

 cat >/etc/docker/daemon.json <<EOF

 {

  "storage-driver": "devicemapper",

  "storage-opts": [

    "dm.thinpooldev=/dev/mapper/docker-
thinpool",

    "dm.use_deferred_removal=true",

    "dm.use_deferred_deletion=true"

    ]

  }

  EOF
```

The map_max_count determines the maximum number of memory map areas a process can have. Docker requires that the max_map_count be substantially greater than the default (65530). For this RA, we have set this parameter in the template:

```
echo "vm.max_map_count=262144" >> /etc/
sysctl.conf
```

Configure password-less Secure Shell (SSH) for root from boot-master to all nodes, including boot-master (assuming a root install). For non-root install, the user must have "no password" sudo privileges to all commands.

With the configured VM, you can now export to the template in vCenter, for later use by Terraform* in the deployment process.

2. Create an RHEL 7.5 boot node VM.

   To simplify installation and configuration of IBM Cloud Private, we have created a separate VM from the previously created template in this RA, which will serve as a boot node.

3. Copy VMs from the template and configure VMs for IBM Cloud Private.

   Use the prepared template VM to create new VMs that you will use as IBM Cloud Private nodes; you will need to configure their vCPU, memory, and disk resources. We strongly recommend using automation tools; at scale, automation will significantly ease and speed up your deployment process.

   To deploy IBM Cloud Private in a large deployment with HA, this RA uses Terraform scripts from the IBM Cloud Architecture GitHub*: github.com/ibm-cloud-architecture/terraform-icp-vmware.

   The RA uses the VMware vSphere provider to provision and configure VMs on the vSphere stack and deploy IBM Cloud Private on top of them. For this RA, we used this repository only to deploy VMs; after deploying the VMs, continue the installation by using the IBM documentation: ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/installing/install_containers.html

4. Download and load the IBM Cloud Private Docker images on the boot machine.

   Download the IBM Cloud Private 3.1.1 for Linux (x86_64) Docker package from IBM Passport Advantage (ibm.com/software/passportadvantage/) onto the boot node and then extract the images and load them into the Docker:

   ```
   tar xf ibm-cloud-private-x86_64-3.1.1.tar.gz
   -O | sudo docker load
   ```

   Create the installation directory and extract the configuration file from the installer image:

   ```
   sudo mkdir /opt/ibm-cloud-private-3.1.1

   cd /opt/ibm-cloud-private-3.1.1

   sudo docker run -v $(pwd):/data -e
   LICENSE=accept \ibmcom/icp-inception-
   amd64:3.1.1-ee \cp -r cluster /data
   ```

5. Prepare the IBM Cloud Private configuration file, *config.yaml*.

   The IBM Cloud Private configuration file incorporates all IBM Cloud Private cluster configuration settings, even when nonexpendable for a deployment process. Below are additional parameter settings provided for installation customization. The rest of the parameters and values are left as default; these are collected in the official IBM Cloud Private 3.1.1 documentation: ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/installing/config_yaml.html.

   Network management is set as Calico* value: *network_type: calico*

   The IPv4 network consistent with Classless Inter-Domain Routing (CIDR) format is set to the entire network using *network_cidr: 10.1.0.0/16*

The Kubernetes service cluster IP range, which is a virtual network also in CIDR format, allocates a block of reserved IPs for cluster services: *service_cluster_ip_range: 10.0.0.0/16*

Kubelet requires additional argument configuration for disabling swap. It is used to pack instances as tightly as possible in the utilized environment. Moreover, the scheduler shouldn't use swap until cluster performance decreases: *kubelet_extra_args: ["--fail-swap-on=false"]*

The following etcd parameters can disable additional wait duration before closing a nonresponsive connection, disable the frequency duration to ping if a server-to-client connection is still alive, and determine the number of committed transactions for which to do a disk snapshot: *etcd_extra_args: ["--grpc-keepalive-timeout=0", "--grpc-keepalive-interval=0", "--snapshot-count=10000"]*

The following commands set cluster-administrator privilege (name and password):

```
default_admin_user: admin
default_admin_password: admin
```

Choose the service that will be managing the virtual IP (VIP) in the case of an HA environment on any combination of master and proxy nodes: *vip_manager: keepalived*

Two of the master-node HA settings for setting the correct interface and virtual IP address in the IBM Cloud Private HA cluster are *vip_iface: ens192* and *cluster_vip: 172.17.255.240*

Two of the proxy-node HA settings for setting the virtual IP interface and address for a proxy node in the IBM Cloud Private HA environment are *proxy_vip_iface: ens192* and *proxy_vip: 172.17.255.241*

Cloud-provider settings and configuration files for VMware Cloud Providers* were chosen because of compatibility with VMware components (vCenter especially), in addition to their interface format, which is POSIX compliant. It is worth remembering the prerequisites before setting any storage options; these are described in detail for VMware Cloud Providers in the official documentation for IBM Cloud Private v3.1.1: ibm.com/support/knowledgecenter/SSBS6K_3.1.1/manage_cluster/vsphere_prereq.html. The user that is indicated in the vSphere cloud configuration file (user: "administrator@vsphere.local") requires privileges to interact with vCenter:

```
kubelet_nodename: hostname
cloud_provider: vsphere

vsphere_conf:
  user: "administrator@vsphere.local"
  password: <type-your-password-here>
  server: vcenter.icp.igk
  port: 443
  insecure_flag: 1
  datacenter: ICP-dc
  datastore: vsanDatastore
  working_dir: ICP-dc
monitoring:
 mode: managed
 prometheus:
   scrapeInterval: 1m
   evaluationInterval: 1m
   retention: 24h
   persistentVolume:
     enabled: false
     storageClass: "-"
   resources:
     limits:
       cpu: 500m
       memory: 8192Mi
 alertmanager:
   persistentVolume:
     enabled: false
     storageClass: "-"
   resources:
     limits:
       cpu: 200m
       memory: 256Mi
 grafana:
   user: "admin"
   password: "admin"
   persistentVolume:
     enabled: false
     storageClass: "-"
   resources:
     limits:
       cpu: 500m
       memory: 2048Mi
```

6. Prepare /etc/hosts, sharing it between all IBM Cloud Private nodes, and prepare the Ansible* host file.

   For /etc/hosts, we have added the IP addresses and host names for all nodes in the cluster, commenting out the lines that begins with 127.0.1.1 and ::localhost.

   ```
   <master_nodes_IP_address> <master_nodes_
   host_name> <worker_nodes_1_IP_address>
   <worker_nodes_1_host_name> <proxy_nodes_
   IP_address> <proxy_nodes_host_name>

   <management_nodes_IP_address> <management_
   nodes_host_name>

   <va_nodes_IP_address> <va_nodes_host_name>
   ```

   This file needs to be shared across all IBM Cloud Private nodes and updated if you add more nodes in the future.

   The Ansible hosts were prepared according to IBM Cloud Private documentation (ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/installing/hosts.html) and copied to the cluster folder within the boot-node VM. We strongly recommend using automation tools in order to ease the deployment process and allow for faster preparation to scale out your IBM Cloud Private cluster.

7. Start the IBM Cloud Private Ansible installer in the Docker container on the boot node.

   Before starting the installation process, it is important to share SSH keys among cluster nodes. Per IBM Cloud Private documentation (ibm.com/support/knowledgecenter/SSBS6K_3.1.1/installing/ssh_keys.html), you must generate the SSH key on a boot node and share the public key among all IBM Cloud Private nodes:

   ```
   ssh-copy-id -i ~/.ssh/id_rsa.pub
   <user>@<node_ip_address>
   ```

   And copy the private key to the cluster folder:

   ```
   sudo cp ~/.ssh/id_rsa ./cluster/ssh_key
   ```

   By default, the command to deploy your environment is set to deploy 15 nodes at a time. If your cluster has more than 15 nodes, the deployment might take longer to finish. If you want to speed up your deployment, you can specify a higher number of nodes to be deployed at a time. To do so, use the argument *-f <number of nodes to deploy>* with the command:

   ```
   sudo docker run --net=host -t -e
   LICENSE=accept \

   -v "$(pwd)":/installer/cluster ibmcom/icp-
   inception-amd64:3.1.1-ee install –vv –f 23
   ```

8. Run a health check after the installation to ensure all services are running properly.

   After a successful deployment, the access information for your cluster will be displayed:

   ```
   UI URL is https://<ip_address>:8443 ,
   default username/password is admin/admin
   ```

   We recommend that you also run a health check to ensure that all services are up and running:

   ```
   sudo docker run --net=host -t -e
   LICENSE=accept -v "$(pwd)":/installer/
   cluster ibmcom/icp-inception-amd64:3.1.1-ee
   healthcheck –vv –f 23
   ```

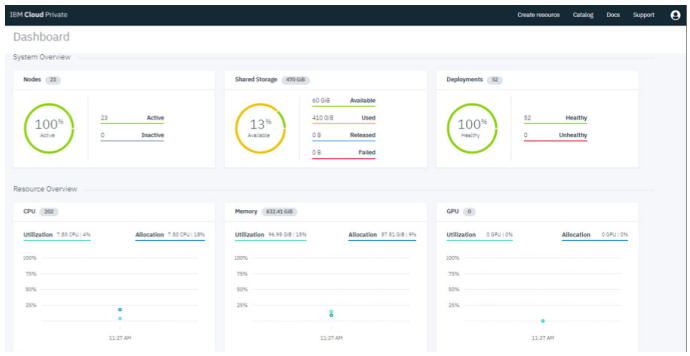


Figure 5. IBM Cloud Private* dashboard after a successful deployment and health check

Networking Details

Use the following section to configure the network environment for your Intel Reference Solution for IBM Cloud Private deployment. In the tables under Network Architecture, we present a sample deployment using two NICs on each server (Tables 9 and 10).

Network Architecture

Table 9. Intel® Reference Solution for IBM Cloud Private* recommended network addresses

VLANS TO CONFIGURE ON REDUNDANT SWITCHES	SERVER PORTS	SAMPLE VLAN SAMPLE IP RANGE	CIDR	VMWARE VSPHERE* DISTRIBUTED SWITCH
IBM Cloud Private*	NIC1, NIC2	Untagged	100.65.80.0/22	DPortGroup-ICP
VMware ESXi* (management)	NIC1, NIC2	Untagged	100.65.80.0/22	DPortGroup-MGMT
VMware vSAN*	NIC1, NIC2	11	192.168.8.0/24	DPortGroup-vSAN
VMware vSphere vMotion*	NIC1, NIC2	12	192.168.12.0/24	DPortGroup-vMotion

Table 10. Details for all networks on top of VMware vSphere* using VMware vSphere Distributed Switch* and four separate distributed port groups

VLANS TO CONFIGURE ON REDUNDANT SWITCHES	SERVER PORTS	SAMPLE VLAN	SAMPLE IP RANGE	VMWARE VSPHERE* DISTRIBUTED SWITCH
IBM Cloud Private* (VMs)	NIC1, NIC2	Untagged	172.17.255.0/24	DPort Group: ICP
VMware ESXi* (management)	NIC1, NIC2	Untagged	172.17.255.0/24	DPort Group: MGMT
VMware vSAN*	NIC1, NIC2	11	192.168.8.0/24	DPort Group: vSAN
VMware vSphere vMotion*	NIC1, NIC2	12	192.168.12.0/24	DPort Group: vMotion
ADDITIONAL NETWORKS				
Hardware network for Intelligent Platform Management Interface (IPMI) management		192.168.24.0/24		
IBM Cloud Private Calico* networks on top of the IBM Cloud Private (VMs) network (overlay):		10.1.0.0/16		
•IBM Cloud Private Calico network for pods		10.0.0.0/16		
•IBM Cloud Private Calico network for services				

Management Network

The management network uses a separate network and cards because it has no dependencies to the other networks in the Intel Reference Solution for IBM Cloud Private. You can use your organizations' own best practices and processes in provisioning it.

Monitoring, Logging, and Metering in IBM Cloud Private

The IBM Cloud Private software platform provides all necessary tools for monitoring, logging, metering, and alerting. The platform makes it easy to manage processes without specialized expertise.

Logging

The ELK* stack in IBM Cloud Private combines Elasticsearch*, Logstash*, and Kibana* into one package to use with IBM Cloud Private for logging Kubernetes and Docker events. You can deploy ELK from the IBM Cloud Private catalog to one of the namespaces in the cluster in order for it to log messages from Docker applications.

Monitoring and Alerting

IBM Cloud Private uses Prometheus* and Grafana* to gather and visualize data from the cluster and applications. Prometheus or add-on tools can be used to provide alerts, with or without notifications.

The Grafana dashboard, which is part of the cluster monitoring dashboard provided by IBM Cloud Private, offers fourteen different types of dashboard templates to monitor the status of applications deployed through the cluster. Examples include:

- **Kubernetes Pod Overview**, which contains pod-specific metrics, such as CPU, memory, and network status and how many restarts single pod has had.

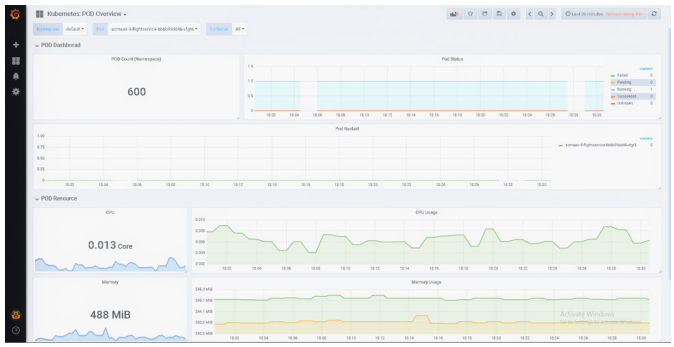


Figure 6. IBM Cloud Private* Grafana* dashboard—Kubernetes* POD overview

- **Kubernetes Cluster Monitoring**, which provides information collected by Prometheus about the Kubernetes cluster and includes CPU, memory, and file system usage. Moreover, the dashboard shows essential information about components, such as individual pods, containers, and system services.

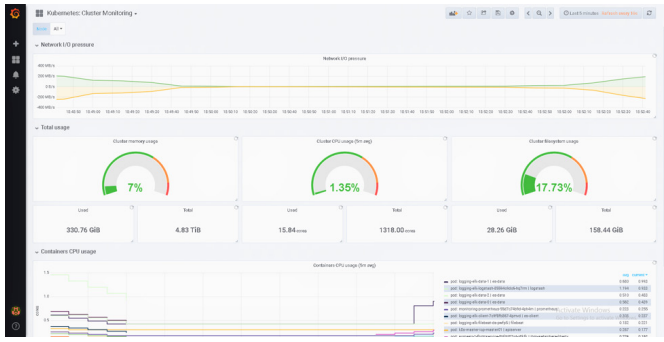


Figure 7. IBM Cloud Private* Grafana* dashboard—Kubernetes* cluster monitoring

- **NGINX Ingress Controller***, which includes an array of metrics about the ingress controller, including requests, volume, and network I/O pressure.

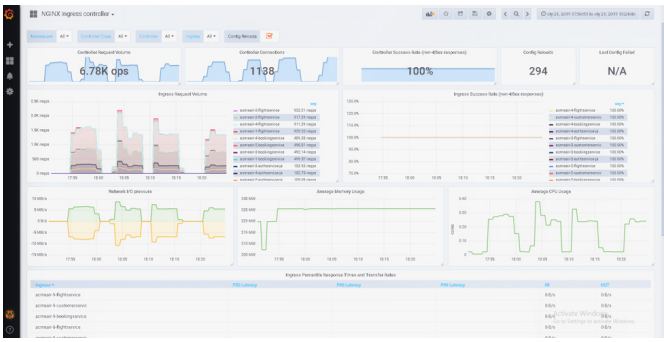


Figure 8. IBM Cloud Private* Grafana* dashboard—NGINX Ingress Controller*

Metering

Metering is provided by IBM Cloud Private for tracking application and cluster metrics.

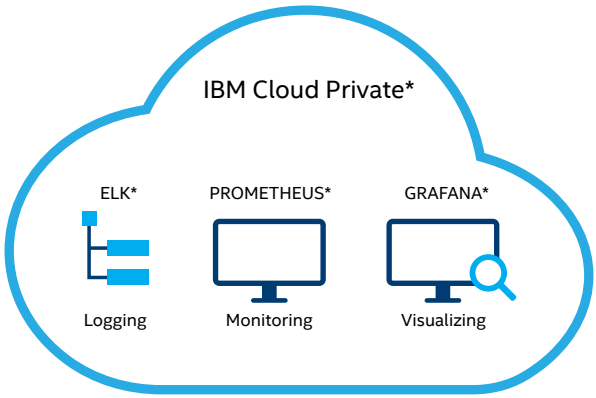


Figure 9. Logging and monitoring tools in IBM Cloud Private*

Network Configuration

The default network for IBM Cloud Private pods is provided by Calico, which is configured automatically during the installation process of IBM Cloud Private on top of the network configured for VMs in vSphere.

Calico is a new approach to virtual networking and network security for containers, VMs, and bare-metal services. It provides a rich set of security-enforcement capabilities that run on a highly scalable and efficient virtual network. The Calico node runs in a Docker container on the Kubernetes master node and on each Kubernetes worker node in the cluster. The calico-cni plugin integrates directly with the Kubernetes kubelet process on each node to discover which pods are created and add them to Calico networking.

For this RA's setup, Calico is configured with IP-in-IP encapsulation. You must set this parameter even if all the nodes belong to the same subnet. This configuration enables encapsulation of pod-to-pod traffic over the underlying network infrastructure.

The calico_tunnel_mtu parameter must be set based on the maximum transmission unit (MTU) of the interface that is configured for use by Calico.

If the IP-in-IP encapsulation parameter is set to true, 20 bytes are used for the IP-IP tunnel header. You must set the calico_tunnel_mtu parameter to be at least 20 bytes less than the actual MTU of the interface.

IBM Cloud Private Calico networks on top of the IBM Cloud Private VMs network overlay:

- IBM Cloud Private Calico network for pods, 10.1.0.0/16—The IPv4 network to use for the entire pod network.
- IBM Cloud Private Calico network for services, 10.0.0.0/16—The Kubernetes service cluster IP range. This configuration allocates a block of IPs for services. These service IPs do not need to be routable, since kube-proxy converts service IPs to pod IPs before traffic leaves the node.

The IBM Cloud Private network for pods' underlying network for VMs and the IBM Cloud Private service cluster IP range must not be in conflict with each other.

Three system-configuration parameters on IBM Cloud Private nodes and the ingress controller must be set to support a large number of open files and TCP connections with large bursts of messages. Changes can be made using the /etc/rc.d/rc.local or /etc/sysctl.conf script to preserve changes after reboot.

1. /proc/sys/fs/file-max: The maximum number of concurrently open files.
2. /proc/sys/net/ipv4/tcp_max_syn_backlog: The maximum number of remembered connection requests, which did not receive an acknowledgment from the connecting client. The default value is 1,024 MB for systems with more than 128 MB of memory, and 128 MB for low-memory machines.
3. /proc/sys/net/core/somaxconn: Limit of socket listen() backlog, known in user space as SOMAXCONN. Defaults to 128.
4. /proc/sys/net/ipv4/ip_local_port_range: Increase system IP port limits to allow for more connections.
5. net.ipv4.ip_local_port_range = 1024 65535

Large numbers of workloads running concurrently can create problems on Linux systems—particularly for proxy nodes—concerning running out of space in the conntrack table, which can cause poor iptables performance. To avoid these issues, we increased the maximum number of tracked connections on the proxy nodes:

```
echo 524288 > /proc/sys/net/netfilter/nf_
conntrack_max
```

Walkthrough: Acme Air*

Acme Air* is a performance-analysis application dedicated to a public cloud environment. It is the main IBM workload that was used for benchmarking cloud-based environments. This fictitious airline application is programmed in Java* and based on a microservices architecture. Microservices

implementations support various types of runtimes, such as bare-metal systems, VMs, or Docker-based platforms. Acme Air covers several key business requirements, including: scalability for dozens of web API calls per day, deployability to a public cloud infrastructure, and the support for a variety of end-user interactions with the system.

There are several ways to deploy the Acme Air application. The implementation described in this RA is of BLUEPERF* containerized applications deployed on IBM WebSphere Liberty* (https://github.com/blueperf/Overview).

Every Acme Air workload contains the following components (five web services and three databases):

1. **acmeair-authservice-java** (generate a JavaScript Object Notation* [JSON] web token if the user/password is valid)
2. **acmeair-bookingservice-java** (store, update, and retrieve booking data)
3. **acmeair-customerservice-java** (store, update, and retrieve customer data)
4. **acmeair-flightservice-java** (retrieve flight-route data)
5. **acmeair-mainservice-java** (presentation layer that interacts with other services)
6. **acmeair-booking-db** (booking information)
7. **acmeair-customer-db** (customer information)
8. **acmeair-flight-db** (flight information)

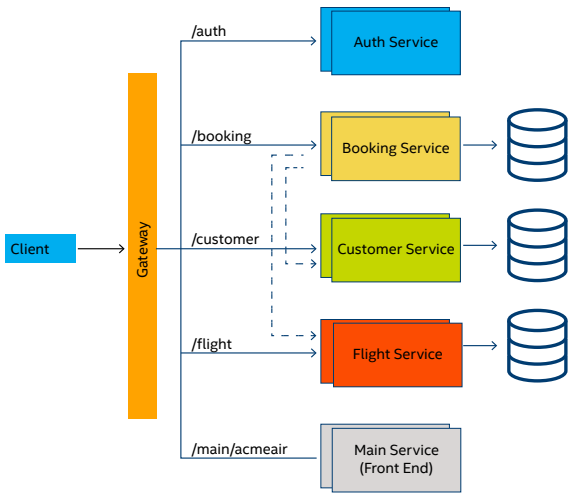


Figure 10. Acme Air* application structure (image source: https://github.com/blueperf/acmeair-mainservice-java/raw/master/images/AcmeairMS.png)

The following steps provide an overview of the setup process.

Step 1: Install required binaries and packages for the IBM Cloud Platform cluster.

1. Install the following binaries, which require CLI tools: kubectl, cloudctl, and Helm*. Then log on to the cluster with the cloudctl command:

```
cloudctl login –a --skip-ssl-validation
```

2. Install the following packages: Docker, IBM Java SDK, or Oracle's Java SDK, Maven*.

Step 2: Configure the IBM Cloud Private cluster.

1. Secondly, add the *docker.io/mongo* image to *ClusterImagePolicies* for properly deploying MongoDB* pods:

```
kubectl edit clusterimagepolicies
$(kubectl get clusterimagepolicies --no-
headers | awk '{print $1}')
```

Edit the deployment configuration file to include the additional entries provided below:

- name: docker.io/mongo

- name: docker.io/mongo:*

- name: docker.io/mongo/mongo:*

2. IBM then recommends performing a copy operation on the Docker certification located on a master node:

```
mkdir /etc/docker/certs.d/<cluster_CA_
domain>:8500/

scp /etc/docker/certs.d/<cluster_CA_
domain>\:8500/ca.crt \

root@<client_node>:/etc/docker/
certs.d/<cluster_CA_domain>\:8500/
```

3. Clone the necessary repositories into the directory you created (for example, */acmeair*):

```
export ACMEAIRDIR=~/acmeair

mkdir -p ~/acmeair

cd ~/acmeair

git clone https://github.com/blueperf/
acmeair-mainservice-java.git

git clone https://github.com/blueperf/
acmeair-flightservice-java.git

git clone https://github.com/blueperf/
acmeair-customerservice-java

git clone https://github.com/blueperf/
acmeair-bookingservice-java

git clone https://github.com/blueperf/
acmeair-authservice-java
```

4. Make sure that the Java variable looks correct and is located in the path. Simple easy-check commands should confirm proper toolchain settings:

```
java –version

mvn -v
```

Step 3: Deploy the Acme Air workload.

1. Build and push Docker images to the IBM Cloud Private registry by executing the *./buildAndPushToICP.sh* script, located in the *acmeair-mainservice-java/scripts* directory:

```
sh ~/acmeair/acmeair-mainservice-java/
scripts/buildAndPushtoICP.sh
```

2. Deploy the application with the *deployChartToICP.sh* script:

```
sh ~/acmeair/acmeair-mainservice-java/
scripts/deployChartToICP.sh
```

Step 4: Start the Acme Air auto-deployment.

1. The Acme Air application can be deployed manually and automatically by running a variation of BLUEPERF scripts. These scripts are prepared to cover scaling aim in the future workflow plan.
2. The automatic script for IBM Cloud Private cluster configuration is used. It includes all of the necessary dependencies, packages, or operations that users should perform manually before the Acme Air deployment process.
3. All of the Docker images should be pushed to the IBM Cloud Private cluster before any other operation.

   a. Put all of the modified Helm scripts and other templates in a specific directory hierarchy:

```
-- <MAIN_DIR>
---- bootstrap_clis.sh
---- deploy_charts.sh
---- db-templates
------ booking-database.yml
------ customer-database.yml
------ flight-database.yml
```

   This group of scripts contains the following changes:

   - In *deploy_charts.sh* (default deployment script is *deployChartToICP.sh* in the official *mainservice-java/scripts* directory) for each service is provided (note that the *name_of_service* variable means main, auth, booking, customer, or flight):

     I. Enable the option for putting a function parameter, which is responsible for changing the names of different deployments. Previously, all of the deployments had exactly the same name:

```
--name="acmeair-(name_of_service)
service-java" \
```

     to

```
--name="acmeair-${1}-(name_of_
service)service-java" \
```

     and

```
--set service.name="acmeair-(name_
of_service)-service" \
```

     to

```
--set service.name="acmeair-${1}-
(name_of_service)-service" \
```

     II. Enable an additional option for modifying the namespace in which the Docker image was built. It is not necessary, but it can be useful in the future.

```
--set image.
repository="${CLUSTER}:8500/default/
acmeair-(name_of_service)service-
java" \
```

     to

```
--set image.repository="mycluster.
icp:8500/${2}/acmeair-(name_of_
service)service-java" \
```

     III. Add a function to ensure access to the application through the virtual host mechanism in NGINX Ingress Controller*, which was used by default in IBM Cloud Private.

```
--set ingress.host="acme${1}.
mycluster.icp" \
```

     IV. Authservice and bookingservice, which have other client-side services, are necessary to change all of the *--set env.jvmArg* parameters.

```
-Dcom.acmeair.client.
CustomerClient/mp-rest/url=http://
acmeair-(name_of_service)-
service:9080
```

     to

```
-Dcom.acmeair.client.
CustomerClient/mp-rest/url=http://
acmeair-${1}-(name_of_service)-
service:9080
```

     This allows deploying "n" numbers of standalone services without resolving name conflicts in the cluster space.

     V. A similar operation involves services connecting with databases, which are booking, customer, and flight. In this case, the *--set env.jvmArgs* parameter is changed from:

```
-DMONGO_HOST=acmeair-(name_of_
service)-db
```

     to

```
-DMONGO_HOST=acmeair-${1}-(name_
of_service)-db
```

   - In the db-templates directory, which contains *booking-database.yaml*, *customer-database.yaml*, and *flight-database.yaml* files for each database configuration file, change:

```
service: acmeair-flight-db
service: acmeair-customer-db
service: acmeair-booking-db
```

     to

```
service: "{{ .Values.db.name }}"
```

     Based on BLUEPERF scripts from:

     - https://github.com/blueperf/acmeair-bookingservice-java/blob/master/chart/ibm-websphere-liberty/templates/booking-database.yaml
     - https://github.com/blueperf/acmeair-customerservice-java/blob/master/chart/ibm-websphere-liberty/templates/customer-database.yaml

- https://github.com/blueperf/acmeair-flightservice-java/blob/master/chart/ibm-websphere-liberty/templates/flight-database.yaml

4. Replace each of the default templates with modified ones to make an opportunity for deploying databases with services by using different names:

   a. After necessary script modification, described in the steps above, and setting a specific directories structure, run autodeployment by executing:

      ```
      ./deploy_charts.sh <num_of_deployments>
      <namespace>
      ```

      Where:

      `<num_of_deployments>`: Number of deployments that are created (one deployment contains five web services and three databases)

      `<namespace>`: The option to specify space, where deployment should be located

5. Deployments will be available on *acme(1..n).mycluster.icp* virtual hostnames, which are resolving to the IBM Cloud Private proxy IP and served by NGINX Ingress Controller.

   a. To finalize the deployment process, as BLUEPERF repository suggests, load the databases with a generated variety of random data. This is achieved by executing the *load_databases.sh* script provided below:

      ```
      for i in `seq $1`

      do

      curl http://acme${1}.mycluster.icp/booking/
      loader/load -L -k

      curl http://acme${1}.mycluster.icp/flight/
      loader/load -L -k

      curl http://acme${1}.mycluster.icp/customer/
      loader/load?numCustomers=10000 -L -k

      done
      ```

   b. Run the following command:

      ```
      run like ./load_databases.sh N
      ```

      Where *N* specifies the number of deployed application instances.

6. IBM Cloud Private dashboard: After the installation process, it is possible to change deployed application services by editing deployment code via dashboard provided by the IBM Cloud Private platform.

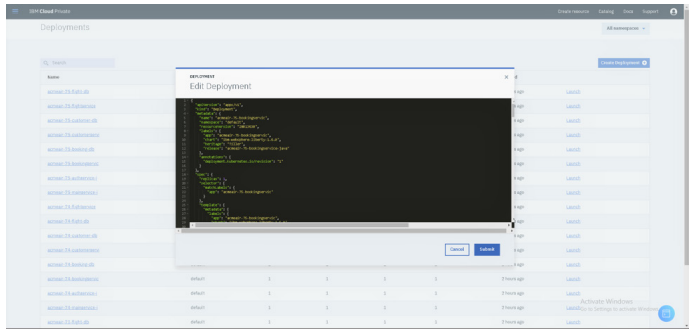


Figure 11. IBM Cloud Private* dashboard—deployments configuration panel

Summary and Conclusions

Deploying the Intel Reference Solution for IBM Cloud Private provides an integrated cloud solution in which you can run VMs and containerized applications side by side. This RA provided prescriptive guidance on how to prepare, provision, deploy, and manage an IBM Cloud Private solution on VMware vSAN ReadyNodes that utilizes Intel hardware and technologies with the performance and security necessary to provide an economical, enterprise-grade way forward for digital transformation.

Addendum: Acme Air Scripts

deploy_charts.sh

```
CLUSTER_IP=172.17.255.240

CLUSTER_PORT=8443

CLUSTER_ADDR=${CLUSTER_IP}:${CLUSTER_PORT}

CLUSTER_CLIU=${CLUSTER_ADDR}/api/cli

# CLIU is for “CLI URL”, wanted to keep <=4 letter

# shortcuts

NAMESPACE=$2


function deploy_named_chart_to_icp {

  # first parameter tells us how we will modify the
name

  echo $1

  # second parameter sets namespace

  echo $2


  cd “$(dirname “$0”)”

  pwd


 helm install \

    --tls \

    --name=”acmeair-${1}-mainservice-java” \

    --set image.repository=”mycluster.icp:8500/${2}/
acmeair-mainservice-java” \
```

```
    --set image.pullPolicy="Always" \
    --set service.type="ClusterIP" \
    --set service.port="9080" \
    --set service.targetPort="9080" \
    --set service.name="acmeair-${1}-main-service" \
    --set ssl.enabled="false" \
    --set ingress.enabled="true" \
    --set ingress.path="/acmeair" \
    --set ingress.rewriteTarget="/acmeair" \
    --set ingress.host="acme${1}.mycluster.icp" \
    --set env.SECURE_USER_CALLS=false \
    ../chart/ibm-websphere-liberty


  helm install \
    --tls \
    --name="acmeair-${1}-authservice-java" \
    --set image.repository="mycluster.icp:8500/${2}/
acmeair-authservice-java" \
    --set image.pullPolicy="Always" \
    --set service.type="ClusterIP" \
    --set service.port="9080" \
    --set service.targetPort="9080" \
    --set service.name="acmeair-${1}-auth-service" \
    --set ssl.enabled="false" \
    --set ingress.enabled="true" \
    --set ingress.path="/auth" \
    --set ingress.rewriteTarget="/" \
    --set ingress.host="acme${1}.mycluster.icp" \
    --set env.SECURE_USER_CALLS=false \
    --set env.jvmArgs="-Dcom.acmeair.client.
CustomerClient/mp-rest/url=http://acmeair-${1}-
customer-service:9080" \
    ../../acmeair-authservice-java/chart/ibm-websphere-
liberty


  helm install \
    --tls \
    --name="acmeair-${1}-bookingservice-java" \
    --set image.repository="mycluster.icp:8500/${2}/
acmeair-bookingservice-java" \
    --set image.pullPolicy="Always" \
    --set service.type="ClusterIP" \
    --set ssl.enabled="false" \
    --set service.port="9080" \
    --set service.targetPort="9080" \
    --set service.name="acmeair-${1}-booking-service"
\
    --set ingress.enabled="true" \
    --set ingress.path="/booking" \
    --set ingress.rewriteTarget="/" \
    --set ingress.host="acme${1}.mycluster.icp" \
    --set db.name="acmeair-${1}-booking-db" \
    --set resources.constraints.enabled="true" \
    --set resources.limits.cpu="8000m" \
    --set resources.limits.memory="4Gi" \
    --set resources.requests.cpu="3500m" \
    --set resources.requests.memory="1Gi" \
    --set env.SECURE_USER_CALLS=false \
    --set env.jvmArgs="-Dcom.acmeair.client.
CustomerClient/mp-rest/url=http://acmeair-${1}-
customer-service:9080 -Dcom.acmeair.client.
FlightClient/mp-rest/url=http://acmeair-${1}-flight-
service:9080 -DMONGO_HOST=acmeair-${1}-booking-db" \
    ../../acmeair-bookingservice-java/chart/ibm-
websphere-liberty


  helm install \
    --tls \
    --name="acmeair-${1}-customerservice-java" \
    --set image.repository="mycluster.icp:8500/${2}/
acmeair-customerservice-java" \
    --set image.pullPolicy="Always" \
    --set service.type="ClusterIP" \
    --set ssl.enabled="false" \
    --set service.port="9080" \
    --set service.targetPort="9080" \
    --set service.name="acmeair-${1}-customer-service"
\
    --set ingress.enabled="true" \
    --set ingress.path="/customer" \
    --set ingress.rewriteTarget="/" \
    --set ingress.host="acme${1}.mycluster.icp" \
    --set db.name="acmeair-${1}-customer-db" \
    --set env.SECURE_USER_CALLS=false \
    --set env.jvmArgs="-DMONGO_HOST=acmeair-${1}-
customer-db" \
    ../../acmeair-customerservice-java/chart/ibm-
websphere-liberty
```

```
  helm install \
    --tls \
    --name="acmeair-${1}-flightservice-java" \
    --set image.repository="mycluster.icp:8500/${2}/
acmeair-flightservice-java" \
    --set image.pullPolicy="Always" \
    --set service.type="ClusterIP" \
    --set ssl.enabled="false" \
    --set service.port="9080" \
    --set service.targetPort="9080" \
    --set service.name="acmeair-${1}-flight-service" \
    --set ingress.enabled="true" \
    --set ingress.path="/flight" \
    --set ingress.rewriteTarget="/" \
    --set ingress.host="acme${1}.mycluster.icp" \
    --set db.name="acmeair-${1}-flight-db" \
    --set env.SECURE_USER_CALLS=false \
    --set env.jvmArgs="-DMONGO_HOST=acmeair-${1}-
flight-db" \
    ../../acmeair-flightservice-java/chart/ibm-
websphere-liberty
}

function patch_with_recent_chart {
  git clone https://github.com/IBM/charts.git

  ALL_SERVICES=(booking flight customer main auth)

  for service in ${ALL_SERVICES[*]}; do
    cp -rf charts/stable/ibm-websphere-liberty
acmeair-${service}service-java/chart
  done
}

function generate_patch {
  echo "
  metadata:
    annotations:
     ingress.kubernetes.io/ssl-redirect: \"false\"
     nginx.ingress.kubernetes.io/ssl-redirect:
\"false\"
     nginx.ingress.kubernetes.io/connection-proxy-
header: \"keep-alive\"
     ingress.kubernetes.io/connection-proxy-header:
\"keep-alive\" "
}

function disable_ssl {
  for i in `seq $1`; do
    YML=$(generate_patch);
    kubectl patch ingress $( kubectl get ingress -o
name | grep acmeair-${i} | cut -d '/' -f 2 ) -p "$YML"
--type='merge' ;
  done
}

function deploy_charts {

  echo "Num apps: $1"
  echo "Namespace: $2"
  echo "Wrapper starting dir: $3"

  DB_SERVICES=(booking flight customer)
  ALL_SERVICES=(booking flight customer main auth)

  cd $ACMEAIRDIR

  for service in ${ALL_SERVICES[*]}; do
    git clone https://github.com/blueperf/acmeair-
${service}service-java.git
  done

  patch_with_recent_chart

  for service in ${DB_SERVICES[*]}; do
    cp ${3}/db-templates/${service}-database.yaml
acmeair-${service}service-java/chart/ibm-websphere-
liberty/templates/
  done

  cd acmeair-mainservice-java/scripts/

  for i in `seq $1`
  do
    #. ${START_DIR}/deploy_named_chart_to_icp.sh ${i}
$2
    deploy_named_chart_to_icp ${i} $2
    LINE="${CLUSTER_IP} acme${i}.mycluster.icp"
```

```
    LINE_CHECK=`grep -F "$LINE" /etc/hosts | wc -l`
    echo "Check: $LINE_CHECK"
# enable this to make the app automatically add hostnames to /etc/hosts
#    if [ $LINE_CHECK -eq 0 ]; then
#      #echo "$LINE" >> /etc/hosts
#    fi
  done

  cd $START_DIR
}

function load_databases {
  for i in `seq $1`
  do
    curl http://acme${1}.mycluster.icp/booking/loader/load -L -k
    curl http://acme${1}.mycluster.icp/flight/loader/load -L -k
    curl http://acme${1}.mycluster.icp/customer/loader/load?numCustomers=10000 -L -k
  done
}

START_DIR=`pwd`

ACMEAIRDIR=`mktemp -d`
echo $ACMEAIRDIR
cd $ACMEAIRDIR

echo "NUM REPLICAS: $1"
echo "NAMESPACE: $2"

deploy_charts $1 $2 $START_DIR
disable_ssl $1

echo "Sleeping until pods are ready (should be polling containers for creation instead)"
sleep 90
load_databases $1
```

bootstrap_clis.sh

```
CLUSTER_IP=mycluster.icp
CLUSTER_PORT=8443
CLUSTER_ADDR=${CLUSTER_IP}:${CLUSTER_PORT}
CLUSTER_CLIU=${CLUSTER_ADDR}/api/cli
NAMESPACE=$1

function bootstrap_clis {

 WORKING_DIR=`mktemp -d`
 KUBECTL="${WORKING_DIR}/kubectl"
 HELM_ARCHIVE="${WORKING_DIR}/helm.tar.gz"
 HELM_UNPACK_DIR="${WORKING_DIR}"
 HELM="${HELM_UNPACK_DIR}/linux-amd64/helm"
 CLOUDCTL="${WORKING_DIR}/cloudctl"
 BINARIES_PATH=/usr/local/bin

 curl -kLo ${KUBECTL} https://${CLUSTER_CLIU}/kubectl-linux-amd64
 chmod 755 $KUBECTL
 cp $KUBECTL ${BINARIES_PATH}/kubectl

 curl -kLo ${HELM_ARCHIVE} https://${CLUSTER_CLIU}/helm-linux-amd64.tar.gz
 mkdir -p ${HELM_UNPACK_DIR}
 tar xvf ${HELM_ARCHIVE} -C ${HELM_UNPACK_DIR}
 chmod 755 ${HELM}
 cp $HELM ${BINARIES_PATH}/helm

 curl -kLo ${CLOUDCTL} https://${CLUSTER_CLIU}/cloudctl-linux-amd64
 chmod 755 ${CLOUDCTL}
 cp $CLOUDCTL ${BINARIES_PATH}/cloudctl

 cloudctl login -n $NAMESPACE -a https://${CLUSTER_ADDR} --skip-ssl-validation <<< "admin
 admin
 1
 "
}

bootstrap_clis
```

booking-database.yaml

```
##### Booking Database  #####
apiVersion: v1
kind: Service
metadata:
 creationTimestamp: null
 labels:
  service: “{{ .Values.db.name }}”
 name: “{{ .Values.db.name }}”
spec:
 ports:
 - name: “27017”
  port: 27017
  protocol: TCP
  targetPort: 27017
 selector:
  service: “{{ .Values.db.name }}”
status:
 loadBalancer: {}
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
 creationTimestamp: null
 name: “{{ .Values.db.name }}”
spec:
 replicas: 1
 strategy: {}
 template:
  metadata:
   creationTimestamp: null
   labels:
    service: “{{ .Values.db.name }}”
  spec:
   containers:
   - image: mongo
    name: acmeair-booking-db
    ports:
    - containerPort: 27017
     protocol: TCP
    resources:
     requests:
      memory: “512Mi”
      cpu: “5000m”
     limits:
      memory: “7Gi”
      cpu: “8000m”
   restartPolicy: Always
status: {}
```

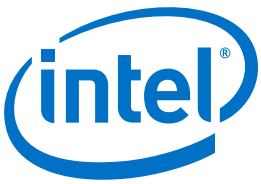
customer-database.yaml

```
##### Customer Database #####
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    service: “{{ .Values.db.name }}”
  name: “{{ .Values.db.name }}”
spec:
  ports:
  - name: “27017”
    port: 27017
    protocol: TCP
    targetPort: 27017
  selector:
    service: “{{ .Values.db.name }}”
status:
  loadBalancer: {}
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  creationTimestamp: null
  name: “{{ .Values.db.name }}”
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
```

```
      labels:
        service: "{{ .Values.db.name }}"
    spec:
      containers:
      - image: mongo
        name: acmeair-customer-db
        ports:
        - containerPort: 27017
          protocol: TCP
      restartPolicy: Always
status: {}
```

flight-database.yaml

```
##### Flight Database #####
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    service: "{{ .Values.db.name }}"
  name: "{{ .Values.db.name }}"
spec:
  ports:
  - name: "27017"
    port: 27017
    protocol: TCP
    targetPort: 27017
  selector:
    service: "{{ .Values.db.name }}"
status:
  loadBalancer: {}
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  creationTimestamp: null
  name: "{{ .Values.db.name }}"
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        service: "{{ .Values.db.name }}"
    spec:
      containers:
      - image: mongo
        name: acmeair-flight-db
        ports:
        - containerPort: 27017
          protocol: TCP
      restartPolicy: Always
status: {}
```



[1] Intel. "Intel® Ethernet Converged Network Adapters X710 DA-2/DA-4." March 2018. intel.com/content/www/us/en/ethernet-products/converged-network-adapters/ethernet-x710-brief.html.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit intel.com/benchmarks.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Cost reduction scenarios described are intended as examples of how a given Intel- based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804

Printed in USA 0219/RM/PRW/PDF Please Recycle 338659-001US