

Running single and scalable containers in IBM Cloud Container Service (Deprecated)

With IBM® Cloud Container Service, manage your apps inside Docker containers, on the IBM cloud. A container is a standard way to package an app and all its dependencies, so that the app can be moved between environments and run without changes. And they have the benefits of resource isolation and allocation, but are more portable and efficient than, for example, virtual machines.

Attention

Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

You can use single and scalable containers to deploy your apps instead of clusters, but with clusters, you have a much wider range of options for customizing your environment. Consider using [clusters](#) as a container orchestration tool instead of single and scalable containers.

For information on managing Docker images for single and scalable containers, see [Getting started with IBM Cloud Container Registry](#).

Migrating single and scalable containers to Kubernetes clusters

You can re-create the single and scalable containers in Kubernetes clusters.

In this page:

- [Re-creating single and scalable container commands in Kubernetes configuration files](#)
 - [Container group example](#)
 - [Single container example](#)

- [Deploying Kubernetes configuration files](#)
- [Migrating auto-scaling](#)
- [Migrating Docker Compose configurations to clusters](#)
- [Removing single containers](#)
- [Removing container groups](#)

Re-creating single and scalable containers commands in Kubernetes configuration files

You can take the properties from single and scalable containers creation commands and create configuration files to deploy similar container pods in Kubernetes clusters. Configuration files are YAML files that contain configuration details for each of the resources you are creating in Kubernetes. Your script might include one or more of the following sections:

Deployment [🔗](#): Defines the creation of pods and replica sets. A pod includes an individual containerized app and replica sets control multiple instances of pods.

Service [🔗](#): Provides front-end access to pods by using a worker node or load balancer public IP address, or a public Ingress route.

Ingress [🔗](#): Specifies a type of load balancer that provides routes to access your app publicly.

[Learn more about the best practices for Kubernetes configuration files.](#) [🔗](#)

Container group example

If you used container groups in the past, you can create a similar setup in a cluster. If you used the VPN service for your container group and you are using a paid account, [you must also order a Vyatta for your cluster's VLAN](#).

You might have created a container group by using the following command. Click a command option to learn how to implement similar functionality in a cluster.

```
bx ic group-create --anti --auto -d AppDomainName -n mycontainerhost
--desired 3 -e CCS_BIND_SRV=my_service_instance -m 1024 --max 5 --min 2
--name my_container_group -p 9080 --volume my_volume:/directory_path
registry.DomainName/ibm/liberty
```

Note

If you used `cf ic group create` command with the `--ip` option, note that existing floating IP addresses cannot be migrated to a Kubernetes cluster. To expose a service on a public IP address, use create a configuration file to deploy a NodePort or LoadBalancer service type. This set up exposes the service on the public IP address of your Kubernetes cluster. [Learn more about public deployment setup options.](#)

--anti

Spread container pods across different worker nodes for higher availability.

```
annotations:
  scheduler.alpha.kubernetes.io/affinity: >
  {
    "podAntiAffinity": {
      "requiredDuringSchedulingIgnoredDuringExecution": [
        {
          "labelSelector": {
            "matchExpressions": [
              {
                "key": "app",
                "operator": "In",
                "values": ["ibmliberty"]
              }
            ]
          },
          "topologyKey": "kubernetes.io/hostname"
        }
      ]
    }
  }
```

--auto

In a standard cluster, use Load balancer or Ingress to enable automatic recovery of

containers. [Learn more about public deployment setup options.](#)

```
type: LoadBalancer
```

-d AppDomainName

In a standard cluster, use Load balancer or Ingress to define a domain. [Learn more about public deployment setup options.](#)

```
type: LoadBalancer
```

-n mycontainerhost

In a standard cluster, use Load balancer or Ingress to define a host. [Learn more about public deployment setup options.](#)

```
type: LoadBalancer
```

--desired 3

To define the desired number of instances of the app, enter a number of replicas.

```
spec:  
  replicas: 3
```

--e CCS_BIND_SRV=my_service_instance

First, [add the service to the cluster](#) by creating a secret. This task is a one-time task that makes the secret for the service available to any app within the cluster. Then, [add that secret and a mount path to the secret](#) in the configuration file for the deployment. All Cloud services that support service keys are supported for use with the Kubernetes functionality. In other words, if you had to use the Cloud Foundry app bridge method for connecting services to single and scalable containers (CCS_BIND_APP), that service is not supported for use with Kubernetes clusters. If you used the direct service binding method (CCS_BIND_SRV), that service is supported for use with Kubernetes.

```
containers:  
  - name: ibmliberty  
    image: "registry.ng.bluemix.net/ibmliberty:latest  
    volumeMounts:  
      - mountPath: /opt/service-bind  
        name: service-bind  
volumes:  
  - name: service-bind
```

```
secret:
  defaultMode: 420
  secretName: service-bind-secret
```

-m 1024

Instead of size, define CPU and memory requests and limits for the container pod.

[Learn more about how to determine CPU and memory values.](#)

```
resources:
  limits:
    cpu: 250m
    memory: 2000Mi
  requests:
    cpu: 125m
    memory: 1000Mi
```

--max 5

Use the HorizontalPodAutoscaler type of configuration file instead of the deployment type that is used in this example. With horizontal auto-scaling, you can set a maximum number of pods. [Learn more about horizontal auto-scaling.](#)

```
maxReplicas: 5
```

--min 2

Use the HorizontalPodAutoscaler type of configuration file instead of the deployment type that is used in this example. With horizontal auto-scaling, you can set a minimum number of pods. [Learn more about horizontal auto-scaling.](#)

```
minReplicas: 2
```

--name *my_container_group*

The name of the resource. You can also assign labels on resources. [Learn more about Kubernetes labels.](#)

```
metadata:
  name: ibmliberty
```

-p 9080

The port to expose to external traffic.

```
ports:
  - containerPort: 9080
```

--volume *volume:/directory_path*

First, [create persistent volume claim](#), and a persistent volume with the requested storage size and storage class is created dynamically for you. Then, [add the claim name and a mount path](#) to the configuration file. To copy data from an existing volume, use the **ibm-backup-restore** image to transfer your data to an IBM Object Storage for IBM Cloud instance.

```
containers:
  - name: ibmliberty
    image: "registry.ng.bluemix.net/ibmliberty:latest"
    volumeMounts:
      - mountPath: /opt/volumemount
        name: myvol
volumes:
  - name: myvol
    persistentVolumeClaim:
      claimName: mypvc
```

registry.DomainName/*ibmliberty*

The image to build the container from. You can continue to use the images in your Cloud image registry, Docker Hub, or your own registry. [Learn more about the registry setup options](#).

```
spec:
  containers:
    - name: ibmliberty
      image: registry.ng.bluemix.net/ibmliberty:latest
```

With the values from the container group example command, use the following configuration file to deploy similar container pods in a cluster:

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: ibmliberty
spec:
  replicas: 3
  template:
```

```
metadata:
  labels:
    app: ibmliberty
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
    {
      "podAntiAffinity": {
        "requiredDuringSchedulingIgnoredDuringExecution": [
          {
            "labelSelector": {
              "matchExpressions": [
                {
                  "key": "app",
                  "operator": "In",
                  "values": ["ibmliberty"]
                }
              ]
            },
            "topologyKey": "kubernetes.io/hostname"
          }
        ]
      }
    }
spec:
  containers:
    - name: ibmliberty
      image: registry.ng.bluemix.net/ibmliberty:latest
      ports:
        - containerPort: 9080
      resources:
        limits:
          cpu: 250m
          memory: 2000Mi
        requests:
          cpu: 125m
```

```

        memory: 1000Mi
    volumeMounts:
    - mountPath: /opt/service-bind
      name: service-bind
    - mountPath: /opt/volumemount
      name: myvol
    volumes:
    - name: service-bind
      secret:
        defaultMode: 420
        secretName: service-bind-secret
    - name: myvol
      persistentVolumeClaim:
        claimName: mypvc
---
apiVersion: v1
kind: Service
metadata:
  name: my-loadbalancer-service
  labels:
    run: ibmliberty
spec:
  selector:
    run: ibmliberty
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 9080

```

Single container example

You might have created a single container by using the following command. Click a command option to learn how to implement similar functionality in a cluster.

```
bx ic run -p 9080 -e CCS_BIND_SRV=my_service_instance -m 1024 --name
```



```
my_container --volume my_volume:/directory_path
registry.DomainName/ibmliberty
```

--e CCS_BIND_SRV=my_service_instance

First, [add the service to the cluster](#) by creating a secret. This task is a one-time task that makes the secret for the service available to any app within the cluster. Then, [add that secret and a mount path to the secret](#) in the configuration file for the deployment. All Cloud services that support service keys are supported for use with the Kubernetes functionality. In other words, if you had to use the Cloud Foundry app bridge method for connecting services to single and scalable containers (CCS_BIND_APP), that service is not supported for use with Kubernetes clusters. If you used the direct service binding method (CCS_BIND_SRV), that service is supported for use with Kubernetes.

```
containers:
  - name: ibmliberty
    image: "registry.ng.bluemix.net/ibmliberty:latest
    volumeMounts:
      - mountPath: /opt/service-bind
        name: service-bind
volumes:
  - name: service-bind
    secret:
      defaultMode: 420
      secretName: service-bind-secret
```

-m 1024

Instead of size, define CPU and memory requests and limits for the container pod. [Learn more about how to determine CPU and memory values.](#)

```
resources:
  limits:
    cpu: 250m
    memory: 2000Mi
  requests:
    cpu: 125m
```

```
memory: 1000Mi
```

--name *my_container_group*

The name of the resource. You can also assign labels on resources. [Learn more about Kubernetes labels.](#)

```
metadata:
  name: ibmliberty
```

-p 9080

The port to expose to external traffic.

```
ports:
  - containerPort: 9080
```

--volume *volume:/directory_path*

First, [create persistent volume claim](#), and a persistent volume with the requested storage size and storage class is created dynamically for you. Then, [add the claim name and a mount path](#) to the configuration file. To copy data from an existing volume, use the **ibm-backup-restore** image to transfer your data to an IBM Object Storage for IBM Cloud instance.

```
containers:
  - name: ibmliberty
    image: "registry.ng.bluemix.net/ibmliberty:latest"
    volumeMounts:
      - mountPath: /opt/volumemount
        name: myvol
volumes:
  - name: myvol
    persistentVolumeClaim:
      claimName: mypvc
```

registry.DomainName/*ibmliberty*

The image to build the container from. You can continue to use the images in your Cloud image registry, Docker Hub, or your own registry. [Learn more about the registry setup options.](#)

```
spec:
  containers:
    - name: ibmliberty
```

```
image: registry.ng.bluemix.net/ibmliberty:latest
```

With the values from the single container example command, use the following configuration file to deploy a similar container pod in a cluster:

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: ibmliberty
spec:
  template:
    metadata:
      labels:
        app: ibmliberty
    spec:
      containers:
        - name: ibmliberty
          image: registry.ng.bluemix.net/ibmliberty:latest
          ports:
            - containerPort: 9080
          resources:
            limits:
              cpu: 300m
              memory: 200Mi
            requests:
              cpu: 200m
              memory: 100Mi
          volumeMounts:
            - mountPath: /opt/service-bind
              name: service-bind
            - mountPath: /opt/volumemount
              name: myvol
      volumes:
        - name: service-bind
          secret:
            defaultMode: 420
```

```
        secretName: service-bind-secret
      - name: myvol
        persistentVolumeClaim:
          claimName: mypvc
---
apiVersion: v1
kind: Service
metadata:
  name: my-loadbalancer-service
  labels:
    run: ibmliberty
spec:
  selector:
    run: ibmliberty
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 9080
```

Deploying Kubernetes configuration files

After the configuration file is created, [you can deploy it to a cluster by using the Kubernetes CLI](#).

Migrating to Kubernetes Autoscaling

With [Kubernetes](#), you can enable [Horizontal Pod Autoscaling](#) to scale your apps based on CPU.

Install the required [CLIs](#).

Create a cluster by following the steps in [Getting started with Kubernetes clusters in IBM Cloud](#). Be sure to set up your cluster to replicate your current container group, including logging, monitoring, vulnerability advisor, and any other customizations you have in place.

[Set the context](#) for your cluster.

Deploy your existing app to your cluster [from the CLI](#). When you deploy your app, you must request CPU.

```
kubectl run <name> --image=<image> --requests=cpu=<cpu> --expose --port=<
```



Understanding this command's components

--image	The application that you want to deploy.
--requests=cpu	The required CPU for your container, which is specified in milli-cores. As an example, --requests=200m.
--expose	When true, creates an external service.
--port	The port where your app is available externally.

Note:

For more complex deployments, you may need to create a [deployment script](#).

Create a Horizontal Pod Autoscaler and define your policy. For more information about working with the `kubectl autoscale` command, see [the Kubernetes documentation](#).

```
kubectl autoscale deployment <deployment_name> --cpu-percent=<percentage>
```



Understanding this command's components

--cpu-percent	The average CPU utilization that is maintained by the Horizontal Pod Autoscaler, which is specified as a percentage.
--min	The minimum number of deployed pods that are used to maintain the specified CPU utilization percentage.
--max	The maximum number of deployed pods that are used to maintain the specified CPU utilization percentage.

Verify that your application is working correctly.

- Check your service dashboard to ensure that your app is running.
- Check your app to be sure that key changes were implemented.

[Deprovision and remove](#) your original instance of Containers.

Migrating Docker Compose configurations to clusters

If you used Docker Compose to deploy your containers, [you can use the Kompose tool from Kubernetes](#) to migrate the Docker Compose configuration files to configuration files that deploy containers into Kubernetes clusters.

Now that you migrated your containers to Kubernetes resources in clusters, check out [Accessing the Kubernetes dashboard](#) to see how those resources are running.

Removing single containers

To maximize the use of your quota, remove containers that are not in use occasionally.

Remove a container by using one of the following methods.

- From the Cloud GUI

From the Cloud Dashboard, select the container that you want to delete.

Expand the **More actions...** menu and click **Delete**.

- In the tile for the container, click the gear icon and click **Delete container**.

- From the CLI *

```
bx ic rm [-f] CONTAINER [CONTAINER]
```

Note

In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk () in this topic.

Optional: Verify that the container was removed by running the following command and confirming that the container does not appear in the list.*

```
bx ic ps -a
```

Removing container groups

To maximize the use of your quota, remove container groups that are not in use occasionally.

Important

Remove the route from the container group by running one of the following commands before you remove a container group or before you delete a route by using the `cf delete-route` command.

```
bx ic route-unmap -n host -d domain group_name_or_ID
```

Remove a container group by using one of the following methods.

- From the Cloud GUI, select your container group and click **Delete** from the **More actions...** menu.

- ```
bx ic group-remove [-f] GROUP [GROUP]
```

Optional: Verify that the container group was removed by running the following command and confirming that the container group does not appear in the list.

```
bx ic groups
```



# About single and scalable containers (Deprecated)

IBM® Cloud provides the IBM Cloud Container Service infrastructure in select regions as a feature for app development. With containers, you can build your app in any language, with any programming tools. Each container is an isolated and secure app platform. A container can run anywhere: from a development workstation that is running OS X or Windows, a server that is running Ubuntu, a production data center virtual machine that is running Red Hat, or a cloud platform like IBM Cloud.

## Attention

Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

In this page:

- [What is a container?](#)
- [Differences between Docker and IBM Cloud Container Service](#)
- [IBM Cloud Container Service in IBM Cloud Dedicated and Local](#)
- [Tutorials and other resources for learning about IBM Cloud Container Service](#)

## What is a container?

Containers are virtual software objects that include all of the elements that an app needs to run. A container is built from an image, which is a read-only template for creating the container. Each image includes just the app and its dependencies, running as an isolated process on the host operating system. Therefore, a container has the benefits of resource isolation and allocation, but is more portable and efficient than, for example, a virtual machine. Containers help you build high-quality apps, fast.

**Containers are agile**

Containers simplify system administration by providing standardized environments for development and production teams. The engine's lightweight run time enables rapid scale-up and scale-down in response to changes in demand. They help remove the complexity of managing different operating system platforms and underlying infrastructure. Containers help you deploy and run any app on any infrastructure, quickly and reliably.

**Containers are small**

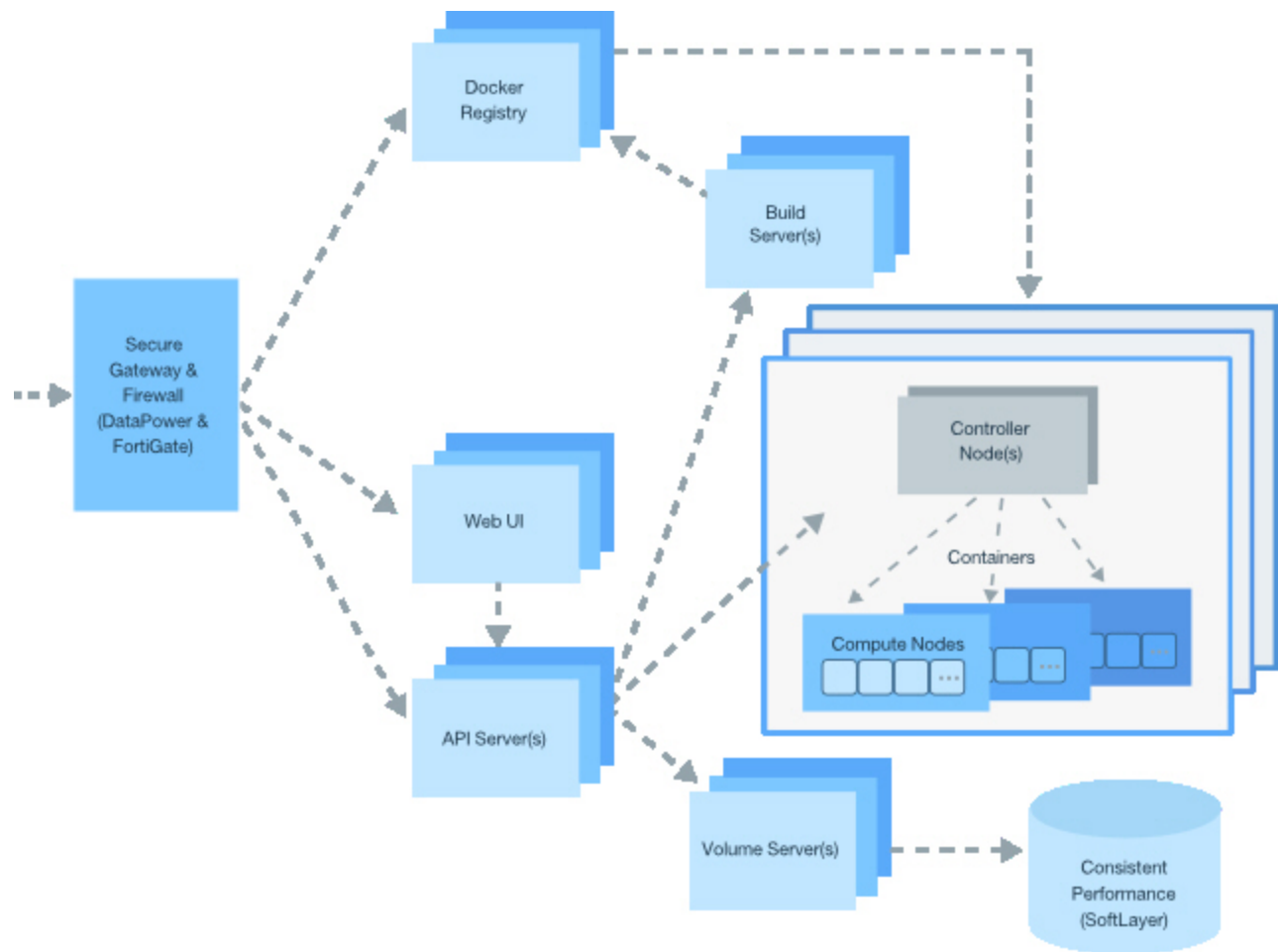
You can fit more containers in the amount of space that a single virtual machine would require. The size of a container can be selected so that you use the amount of space the app requires only.

**Containers are portable**

Build an image for another container by using another image as the base. Let someone else do the bulk of the work on an image and tweak it for your use. You can also migrate app code from a staging environment to a production environment quickly. The migration process can be automated with tools such as the Delivery Pipeline or UrbanCode™ Deploy.

The following figure shows the architecture for IBM Cloud Container Service to run containers in Cloud.

*Figure 1. Architecture overview of IBM Containers*



## Docker and IBM Cloud Container Service

IBM Cloud Container Service is a Cloud runtime that leverages Docker technology. Existing Docker users, and users who are new to Docker, can look forward to the following capabilities with IBM Cloud Container Service.

- Add your existing Docker Hub images to a private registry hosted by IBM.

- Validate images for potential vulnerabilities.

- Configure the command-line interface (CLI) for IBM Cloud Container Service to use the Docker CLI commands that you are comfortable using already.

- See logging and monitoring data for your containers without additional configuration.

If you have used Docker in the past, you might find the answers to these common questions helpful as you get started with IBM Cloud Container Service.

## **How do I use an image from Docker Hub in IBM Cloud Container Service?**

By using the IBM Cloud Container Service CLI, you can [copy an image from Docker Hub](#) to your private Cloud registry. After the image is copied, you can use it to create a container.

## **How do I create a highly available container?**

IBM Cloud Container Service provides single containers that are similar to Docker containers. In addition, IBM Cloud Container Service provides scalable container groups, that can be used to make your application highly available. Container groups provide a built-in load balancer to ensure that the workload is routed and balanced between the container instances. To increase the availability of your group, spread your container group instances across multiple physical compute hosts with anti-affinity and enable auto-recovery of unhealthy instances. Also, you can set policies to auto-scale container group instances based on CPU and memory usage. For more information, see [Running highly available processes as container groups \(Deprecated\)](#).

## **How do I map container ports to a host port?**

IBM Cloud Container Service differentiates between the private container network and the public network. On the private container network, all container ports are exposed by default without additional configuration. Single containers and container groups that are connected to the same private container network, can communicate by using their assigned private IP addresses. No port mapping is necessary.

To make your app available to the public network, you must expose a public port, and bind either a public IP address to your single container, or a public route to your container group. Exposing a public port allows you to send and receive public data on the exposed port only. All other public ports are closed.

## **How do I make a container publicly available on the Internet?**

For [single containers](#), expose one or more ports and assign a floating IP address to the container.

For [container groups](#), expose a single HTTP port and assign a route.

## **What makes the amount of time to build a container different between local Docker and Cloud?**

You might notice that containers build more quickly with your local Docker installation than in Cloud. With the first deployment, the container image must be downloaded to

the registry on the host. Subsequent deployments of the image are faster. For more information about how deployment speeds are impacted by the different container configurations, see [Container types](#).

### **How do I create a container on a private network and not expose it publicly?**

When you create a container, it is connected to the IBM Cloud Container Service default container private network and a private IP address is assigned to your container. On this private network, all container ports are exposed by default. No mapping of host ports to container ports is necessary. To prevent a container from being accessible from the Internet, do not bind an IP address or expose a port when you create the container.

### **How do I mount a volume to a container?**

A single container is, by design, short-lived. However, you can use a volume to persist data between container restarts. When you mount a volume in Docker, the volume is mounted to your local file system. In IBM Cloud Container Service, the access to the compute host is restricted, so you cannot mount host directories to a container. Instead, organization-scoped volumes are used to persist data between container restarts. Volumes are hosted on isolated file shares that securely store app data and manage the access and permission to the files. To mount a volume to a container, you must [create a volume](#) first. When you create a container from the CLI, mount an existing volume by using the `--volume` option. From the Cloud GUI select the **Advanced options** and assign an existing volume. For more information, see [Adding files to volumes with the command line interface \(CLI\)](#).

### **How do I use Docker Compose with IBM Cloud Container Service?**

You can execute Docker Compose commands to create and manage multi-container deployments with IBM Cloud Container Service.

### **How do I set up my Docker compose client so that it points to the IBM Containers service?**

When you log into the IBM Cloud Container Service CLI, configure your CLI run native Docker commands. For more information, see [Logging in to the bx ic plug-in](#).

### **How do I log into my running container? SSH into it?**

You can use `bx ic exec` to [log into a container](#).

### **Why are my containers running in detached mode?**

By default, containers in Cloud run in detached mode to allow containers with running processes to run in the background without exiting. Containers running in detached mode is the standard model for a cloud environment. You can still attach to your container by running `bx ic attach [--no-stdin] [--sig-proxy] CONTAINER`.

# IBM Cloud Container Service in Cloud Dedicated and Cloud Local

IBM Cloud Container Service is available in Cloud Dedicated and Cloud Local.

## Hardware requirements for IBM Cloud Container Service in Cloud Local

### Base hardware requirements

The base setup is configured with 64GB memory of container compute capacity by default.

Your environment must meet the [Cloud Local hardware requirements](#).

To use IBM Cloud Container Service, each setup requires a control plane that meets the following requirements. Two instances are also required for monitoring and logging, meeting the same hardware requirements.

- VMware: 3 ESX servers, each with 256GB memory and 32 physical cores (64 vCore with Hyperthreading enabled). Across the 3 servers, 4TB of cluster shared storage is required.

### Requirements for additional hardware increases

You can increase container compute capacity in 16GB increments up to 256GB without additional hardware by adding up to 12 expansion VMs. Each VM must have 20GB memory, 4 vCPU, and 200GB disk space. However, for each 10 expansion VMs deployed on the cluster, 1 additional VM is required for to handle networking. This networking VM requires 4vCPU, 16GB memory, and 250GB disk space.

## Related links

### Related Links

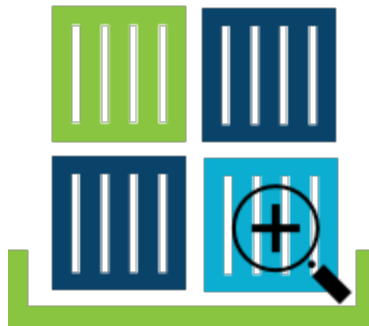
[Cloud Dedicated](#)

[Cloud Local](#)

# Tutorials and other resources for learning about IBM Cloud Container Service

As a supplement to the IBM Cloud Container Service documentation, try out these tutorials and other resources.

## New to Docker



If you never used Docker before, review the Docker user guide. The fundamentals of Docker apply to using containers and images in IBM Containers. On the Docker page, select **Get Started** to install Docker and learn basic Docker and container capabilities.

[Docker user guide](#)

### Audience

Beginners who want to become container developers

### Time required

20 minutes

### Learning Objective

To learn the basics about Docker so that you can apply them to IBM Containers.

## End to end scenarios



Learn containers from installation to deployment with a sample app.

[Docker-based Web Apps, running on IBM Cloud Container Service, Part 1 and Part 2](#)

**Audience**

Developers, admins, and ops managers who are new to IBM Cloud Container Service

**Time required**

Part 1 takes 25 minutes, Part 2 takes 35 minutes

**Learning objective**

To learn about IBM Cloud Container Service and the IBM Cloud architecture and see sample code in IBM Cloud Continuous Delivery.

[Build and extend Docker container images with middleware functions](#)

**Audience**

Container developers and architects

**Time required**

60 minutes

**Learning objective**

To learn how to transition middleware process into Docker containers.

Learn how to use IBM Cloud Continuous Delivery to keep your app and container updated.

[Create and use a secure container toolchain](#)

**Audience**

Container developers

**Time required**

5 minutes



**Learning objective**

To create an open toolchain from a template and use the toolchain to continuously deliver a “Hello World” app to a secure Docker container.

Create and use a simple container toolchain

**Audience**

Container developers

**Time required**

5 minutes

**Learning objective**

To create an open toolchain from a template and use the toolchain to continuously deliver a “Hello World” app to a Docker container.

# Planning to use single and scalable containers (Deprecated)

IBM® Cloud Container Service offers many options to configure your container development environment to meet the functional and non-functional requirements of your app and organization. Learn what the options are before you get started.

## Attention

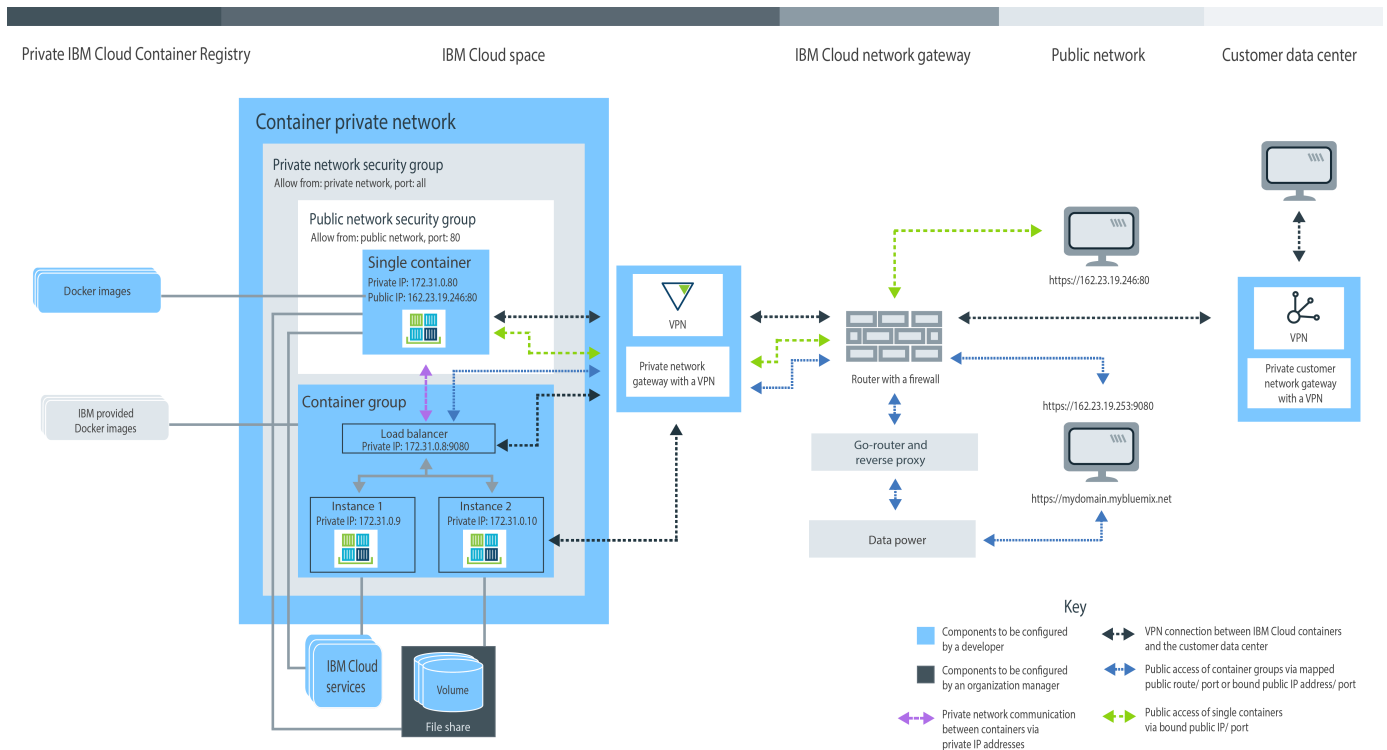
Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

In this page:

- [Understanding user roles](#)
- [Manage your organization for single containers and scalable container groups](#)
  - [Set up a namespace for your private Cloud registry](#)
  - [Quota and available Cloud account types](#)
  - [Persistent storage setup](#)
- [Create single containers and scalable container groups in IBM Cloud Container Service](#)
  - [Cloud GUI, CLI, or REST API](#)
  - [IBM public, Docker Hub, and private images](#)
  - [Private container network settings](#)
  - [Public network settings](#)
  - [Integration of Cloud services](#)
  - [Persistent data storage options](#)
  - [Container types](#)
  - [Monitoring and Logging options](#)

# Understanding user roles

Containers are managed by organization managers and container developers. The following image shows a sample container configuration of an organization and highlights the components that can be set by either the organization manager or the container developer.



Review the sub-sections in this topic to find information about every component, and the decision, and configurations that you can make. Some of these configurations cannot be changed after a container is created. Knowing these configurations in advance can help you to assure that all the resources, such as memory, disk space, and IP addresses are available to the development team as well as to maximize the use of your organization quota.

## Manage your Cloud organization for IBM Cloud Container Service

In Cloud, you can use organizations to enable collaboration among team members and to facilitate the logical grouping of cloud resources in a project. Every organization is assigned an organization manager that is responsible for setting up the development environments (spaces), and to grant team members access to the cloud resources that are needed to successfully create containers for their apps. This set up includes the assignment of public IP addresses,

data storage, and container memory. As an organization manager you can also view the current quota and usage of cloud resources, and adjust them as needed.

Before you begin, [confirm the roles for the users](#) in the organization.

Assign organization managers by giving a user the *manager* role within the organization.

Assign container creators by giving the user a *developer* or *auditor* role within the space.

## Setting your organization's namespace for your private Cloud registry

To store and manage private images for IBM Cloud Container Service, every organization is required to set up its own private Docker images registry in Cloud. This name of the private registry is called a *namespace* and must be unique within Cloud.

The namespace is used to generate a unique URL to your private image registry. Therefore, the namespace is appended to the Cloud registry URL as follows:

`registry.DomainName/namespace`. You must enter the full URL to your private registry whenever you want to work with one of your images from the CLI. If you want to use IBM Cloud Container Service in multiple Cloud regions, you must set up a namespace for every region. All container images that are stored in your private registry are shared by all users of an organization.

### Note

Every organization can have one namespace per Cloud region at a time only. Once the namespace is set for an organization, it cannot be changed, only deleted.

Consider the following rules when you choose a namespace for your organization.

Your namespace must be unique in Cloud.

Your namespace can be 4-30 characters long.

Your namespace must start with at least one letter or number.

Your namespace can only contain lowercase letters, numbers or underscores (\_).

From the Cloud GUI follow these steps.

From the catalog, select **Containers** and choose an image.

If a namespace is already set for your organization, you are not prompted to create one and can continue with the next step. If a namespace is not specified for your organization yet, you are prompted to set one for the image registry.

From the CLI, run the following command after logging in to IBM Cloud Container Service:

Set a namespace for your organization. Replace *<my\_namespace>* with the namespace that you want to set for your organization.

```
bx ic namespace-set <my_namespace>
```

**Note:**

If you are not sure if a namespace was already set for your organization, follow the steps in the [Retrieving the namespace for your organization](#) topic.

## Retrieving the namespace for your organization

If a namespace for your organization is already set, you can retrieve it from the Cloud GUI or the CLI.

### Note

You can identify the namespace from the Cloud GUI only if a private image was already added to your private images registry. You cannot retrieve this information from an image that was provided by IBM. If there are no private images in your registry yet, choose the CLI option for identifying the namespace for your organization.

From the Cloud GUI, follow these steps.

From the catalog, select **Containers** and choose an image.

Under the image name, click **Copy URL**.

Retrieve the namespace from the copied URL, which is in the following format:  
registry.DomainName/*<my\_namespace>*/*<image\_name>*:*<tag>*.

From the CLI, log in to IBM Cloud Container Service service and run the following command.

Retrieve your namespace information.

```
bx ic namespace-get
```

## Quota and Cloud accounts

Every organization in Cloud has a preset value for container memory, public IP addresses, file shares, and the number of services that are shared among all spaces of an organization. These preset values are called quota.

Cloud offers different accounts that you can use with IBM Cloud Container Service. The type of account impacts the quota that is available to your organization and the way you get billed for the usage of the container resources. Any application or service in a space of the organization contributes to the usage of the quota, whether they are running or not. So it is important to know what you can do to [maximize the use of your existing quota](#). The [Pay-As-You-Go or Subscription plans](#) are set up with a [default quota limit](#) that you can [allocate to the Cloud spaces](#) to match the needs of your organization.

### Cloud account types

The following table provides an overview of the account types that are available. For detailed information about each account and billing type, review the [Pricing](#) topic.

Table 1. Cloud account type overview

| Account type  | Description                                                                                                                                                                                |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Free-trial    | Free trials for single and scalable containers are not available. <a href="#">Learn about cluster management and migrate to Kubernetes in IBM Cloud Container Service on Cloud Public.</a> |
| Pay-As-You-Go | If you sign up for a Pay-As-You-Go account, you pay only for the Cloud resources that you use.                                                                                             |
| Subscription  | If you sign up for a Subscription account, you commit to a minimum spending amount each month and receive a subscription discount that is applied to that                                  |

| Account type    | Description                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 | minimum charge. You also pay for any usage that exceeds the minimum spending amount.                                                                                                                                                     |
| Cloud Dedicated | With Cloud Dedicated, you must sign up for a one year minimum term. What you pay each month during that term is based on the dedicated services that you want, plus a subscription account that gives you access to all public services. |
| Cloud Local     | With Cloud Local, you must sign up for a one year minimum term. What you pay each month during that term is based on the local services that you want, plus a subscription account that gives you access to all public services.         |

## Default quota for paid accounts

Every paid account is set up with default quota limits that can be reviewed in the following table. If you require more resources, you can request to add extra quota by opening a [Cloud support ticket](#).

*Table 2. Quota overview for paid accounts*

| Resource            | Paid accounts quota limits                                                                                                                                                                                           |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Public IP addresses | 64                                                                                                                                                                                                                   |
| Container memory    | <div>64 GB</div> <div> <b>Note</b> <p>Every container in your space contributes to the usage of the quota, whether the container is running or not. To free up memory, you must remove unused containers.</p> </div> |
| File shares         | 10 file shares                                                                                                                                                                                                       |
| Containers and      | unlimited number                                                                                                                                                                                                     |

| Resource         | Paid accounts quota limits                                                                                                                                                                                                                                                                                                                          |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| container groups | <div>Note</div> <p>The size of your container and container groups counts towards your container memory limit, but you are not limited to a specific number of containers.</p>                                                                                                                                                                      |
| Images           | 25 <div>Note</div> <p>The IBM images do not count towards your image limit in your private Cloud registry.</p>                                                                                                                                                                                                                                      |
| Spaces           | unlimited number <div>Note</div> <p>Every space requires at least 2 public IP addresses and 2 GB container memory which are automatically allocated when you log into a new space and attempt to use it with the IBM Cloud Container Service. These resources count towards your quota, but you are not limited to a specific number of spaces.</p> |

### Allocating quota to a space

With the pay-as-you-go or subscription plans, you can adjust your quota allocation for memory, number of file shares, and number of public IP addresses for each space until your quota limit is reached. To increase or decrease your space quota for containers, follow these steps.



From your account details, in manage organizations, select an organization.

In the quota section, view the details for containers.

In the **Quota Allocation** table, change the quota that is allocated per space by clicking the **Edit** button.

Save your changes by clicking the **Save** button.

### Tip

If your maximum quota is reached, you can request to add extra quota by opening a [Cloud support ticket](#).

## Tips to maximize the use of your quota

Consider the following tips to release container resources when they are not needed and maximize the use of your quota.

- Make sure to remove containers that are not in use so that their configurations are not counted toward your quota.

- Do not bind a public IP address to your container when your container must not be accessible from the internet.

- To reduce the public IP address usage, consider using container groups in place of a single container. Container groups use a route for public access that is not subject to the IP address quota. For more information , see [Running long-term services as container groups from the Cloud GUI](#).

## Persistent data storage

A file share is a persistent NFS-based (Network File System) storage system that hosts Docker volumes in a Cloud space, and allows a user to store and access container and app-related files. To store files in a file share, you must create a container volume and save the data into this volume.

As soon as you create your first volume in a space, a default file share with 20 GB at 4 IOPS

(Input Output operations Per Second) is created at no cost. The size of the volume is relative to the size of your file share. When adding more volumes, each volume is assigned an equal amount of the file share storage.

The organization manager can create file shares with specific storage size and IOPS to meet the storage needs of the space. File shares can be provisioned in sizes from 20 GB to 12 TB and at IOPS per GB of 0.25, 2 or 4. The file share size in relation to the number of IOPS impacts the speed that data can be read and written from and to the container volume.

To create a file share, see:

[Adding file shares for volumes with the Cloud GUI](#)

[Adding file shares with the command line interface \(CLI\)](#)

## Creating your containers with IBM Cloud Container Service in Cloud

To create containers that run your app with IBM Cloud Container Service in Cloud, you must have been granted developer rights for an organization space. As a developer of apps, it is important to know how container components are related and how you can change the configuration of a container to meet the functional and non-functional requirements of your app.

Before you begin, make sure that you are assigned the *developer* or *auditor* role for the space. The organization manager or owner can grant you access to a space.

### Cloud GUI, the command line, and the REST API

When you are planning to work with IBM Cloud Container Service, you can use the Cloud GUI, install the command line interface (CLI), or send HTTP requests against the REST API to access IBM Cloud Container Service.

The Cloud GUI and the [CLI](#) can be used to complete the majority of tasks in IBM Cloud Container Service. However, there are tasks that can be performed by either the Cloud GUI or the CLI only. So be prepared to use both to manage your containers.

If you neither want to use the Cloud GUI nor the CLI, you can send HTTP requests directly to the [IBM Cloud Container Service API](#) server to manage your containers with REST.

The following table shows all container-related tasks that either require the Cloud GUI, the IBM

Cloud Container Service CLI, or the REST API. Tasks that are not listed here can be performed by all methods.

*Table 3. Container-related tasks that require either the Cloud GUI, IBM Cloud Container Service CLI , or REST API*

| Task                                           | GUI | CLI | API |
|------------------------------------------------|-----|-----|-----|
| Review image vulnerabilities                   | ✓   | ✗   | ✗   |
| Push images to registry                        | ✗   | ✓   | ✓   |
| Pull images from registry                      | ✗   | ✓   | ✗   |
| Copy images from Docker Hub                    | ✗   | ✓   | ✗   |
| Change quota allocation                        | ✓   | ✗   | ✗   |
| Retrieve organization and space specific quota | ✓   | ✗   | ✗   |
| View detailed log information                  | ✓   | ✗   | ✗   |
| Specify more logs                              | ✓   | ✗   | ✗   |
| Customize logs                                 | ✓   | ✗   | ✗   |
| Log into a running container                   | ✗   | ✓   | ✗   |
| Use Docker Compose                             | ✗   | ✓   | ✗   |
| List volumes of a space                        | ✗   | ✓   | ✓   |
| Write files to a volume                        | ✗   | ✓   | ✗   |

| Task                     | GUI | CLI | API |
|--------------------------|-----|-----|-----|
| Delete a volume          | ✗   | ✓   | ✓   |
| Retrieve system messages | ✗   | ✗   | ✓   |

## Installing the CLI

Install and configure your IBM Cloud Container Service CLI to manage your containers from the CLI.

Install the [IBM Cloud Container Service CLI](#) and its prerequisites. To work with IBM Cloud Container Service from the CLI, you must install the Cloud Foundry CLI (cf), the IBM Cloud CLI (bx) and the IBM Cloud Container Service plug-in (bx ic).

Optional: Install [Docker Compose](#). If you want to run Docker Compose commands with IBM Cloud Container Service, additional software packages are necessary.

Log into the IBM Cloud Container Service CLI.

Choose to run IBM Cloud Container Service (bx ic) or native Docker commands.

- **Option 1** (Default)

A simple shell for IBM Cloud Container Service wrappers Docker. The local Docker daemon is unaffected by IBM Cloud Container Service. bx ic commands are used to manage containers in your Cloud space and docker commands are used to manage containers locally. docker commands are always required for local image development though.

For example, with Option 1, the bx ic run command creates a container in your Cloud space and docker run creates a container locally. You might prefer this option if you are new to both Docker and IBM Cloud Container Service or if you like to test containers locally.

- **Option 2**

Redirect your local Docker daemon for the current CLI session to IBM Cloud Container Service so that docker commands can be used in place of bx ic commands whenever they are available to manage containers in your Cloud space.

For example, with Option 2, either the `bx ic run` or the `docker run` command can be used to create a container in your space. You might prefer this option if you are an experienced Docker user who is more comfortable with the `docker` commands or if you are adapting automated scripts to work with IBM Cloud Container Service.

Some exceptions still require the use of `bx ic` commands to perform tasks specific to IBM Cloud Container Service. For example, to set a namespace, you must use the command `bx ic namespace-set`. `docker namespace-set` is not a valid command. For a list of `docker` commands that are supported with this configuration, see [Supported Docker commands for IBM Cloud Container Service plug-in \(bx ic\)](#).

To configure this option, copy and paste the environment variables that are provided in the terminal during log in.

- `DOCKER_HOST` sets the Docker host to IBM Cloud Container Service.
- `DOCKER_CERT_PATH` sets the location of the authentication keys for the IBM Cloud Container Service plug in for Cloud Foundry.
- `DOCKER_TLS_VERIFY` uses TLS and verifies the remote server.

#### Tip

When you are completing the tasks in the IBM Cloud Container Service documentation, you can run `docker` commands in all steps that are marked with an asterisk (\*).

## Managing containers via the REST API

Use the IBM Cloud Container Service API to work with your single containers and container groups using REST.

To log into the IBM Cloud Container Service service, you must log into Cloud first. Follow these steps to retrieve your Cloud access token and to get the ID of the space in which you want to work with your containers.

Run a GET HTTP request against the Cloud API to retrieve the current authentication endpoint information.

```
GET http://api.DomainName/info
```

Your output looks similar to the following.

```
{
 "name": "Cloud",
 "build": "235011",
 "support": "http://ibm.biz/ibmcloud-supportinfo",
 "version": 0,
 "description": "IBM Cloud",
 "authorization_endpoint": "https://login.DomainName/UAALoginServerWAR",
 "token_endpoint": "https://uaa.DomainName",
 "allow_debug": true
}
```

Authenticate with the authorization endpoint to retrieve your Cloud access token.

Locate the **authorization\_endpoint** in the API output of the previous step.

Add **/oauth/token** at the end of the authorization endpoint URL.

Submit a POST request to the authorization endpoint to retrieve your Cloud access token.

```
POST https://login.DomainName/UAALoginServerWAR/oauth/token
```

*Table 4. Request input parameters to authenticate with Cloud*

| Request input parameter | Values                                                                                                                                                                                               |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Header                  | <ul style="list-style-type: none"><li>■ Accept: application/json; charset=utf-8</li><li>■ Authorization: Basic Y2Y6</li><li>■ Content-Type: application/x-www-form-urlencoded; charset=utf</li></ul> |
| Body                    | grant_type=password&username=<username><br>&password=<password>                                                                                                                                      |

| Request input parameter | Values                                                                                                                                 |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
|                         | <div>Note</div> <p>Replace <code>&lt;username&gt;</code> and <code>&lt;password&gt;</code> with your Cloud user name and password.</p> |

Your output looks similar to the following.

```
{
 "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiJkMzh1ZGZiZS0wOTJ",
 "token_type": "bearer",
 "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4YWE3MmI0Yy02MG",
 "expires_in": 1209599,
 "scope": "openid cloud_controller.read password.write cloud_contr",
 "jti": "d38edfbe-092b-482f-90ab-9dc278e4f22d"
}
```

Note the **access\_token** from your CLI response to use it in the next step.

Retrieve the ID of the Cloud space. To authenticate with the IBM Cloud Container Service service, you must have the Cloud access token that you got in the previous step as well as the space ID of your organization, in which you want to work with your containers.

If you have your space ID, then you can skip this step and go to the next step. If you are unsure what your space ID is, follow these steps to retrieve it.

Retrieve the ID of your Cloud organization.

```
GET http://api.DomainName/v2/organizations
```

Table 5. Request input parameters to retrieve a Cloud organization and space

| Request input parameter | Values                                                                                    |
|-------------------------|-------------------------------------------------------------------------------------------|
| Header                  | <ul style="list-style-type: none"> <li>Accept: application/json; charset=utf-8</li> </ul> |

| Request input parameter | Values                                                                                                                                                                     |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                         | <ul style="list-style-type: none"> <li>■ Authorization: bearer &lt;access_token&gt;</li> <li>■ Content-Type:<br/>application/x-www-form-urlencoded; charset=utf</li> </ul> |

Your output looks similar to the following. The ID for your organization can be found as **guid** in the **metadata** section.

```
"metadata": {
 "guid": "31eb6110-5935-446f-ac94-f97f5852b19b",
 "url": "/v2/organizations/31eb6110-5935-446f-ac94-f97f5852b19b",
 "created_at": "2016-01-07T18:55:19Z",
 "updated_at": "2016-02-09T15:56:22Z"
},
"entity": {
 "name": "org@us.ibm.com",
 "billing_enabled": false,
 "quota_definition_guid": "7f66aaf3-9e9b-49f4-83d9-40d92d318",
 "status": "active",
 "quota_definition_url": "/v2/quota_definitions/7f66aaf3-9e9b-49f4-83d9-40d92d318",
 "spaces_url": "/v2/organizations/31eb6110-5935-446f-ac94-f97f5852b19b/spaces",
 "domains_url": "/v2/organizations/31eb6110-5935-446f-ac94-f97f5852b19b/domains",
 "private_domains_url": "/v2/organizations/31eb6110-5935-446f-ac94-f97f5852b19b/private_domains",
 "users_url": "/v2/organizations/31eb6110-5935-446f-ac94-f97f5852b19b/users",
 "managers_url": "/v2/organizations/31eb6110-5935-446f-ac94-f97f5852b19b/managers",
 "billing_managers_url": "/v2/organizations/31eb6110-5935-446f-ac94-f97f5852b19b/billing_managers",
 "auditors_url": "/v2/organizations/31eb6110-5935-446f-ac94-f97f5852b19b/auditors",
 "app_events_url": "/v2/organizations/31eb6110-5935-446f-ac94-f97f5852b19b/app_events",
 "space_quota_definitions_url": "/v2/organizations/31eb6110-5935-446f-ac94-f97f5852b19b/space_quota_definitions"
}
```

Locate the **spaces\_url** field in the **entity** section of your API output from the previous step. Retrieve a list of all the spaces of that organization by running a GET API call against the **spaces\_url**. To run the API call, use the same header



information as you did in the previous step.

```
GET http://api.DomainName/v2/organizations/31eb6110-5935-446f-ac94-
```

Your output looks similar to the following. The ID for your space can be found as **guid** in the **metadata** section.

```
"metadata": {
 "guid": "0e65c436-9413-46e2-8b42-ff06aa74ac21",
 "url": "/v2/spaces/0e65c436-9413-46e2-8b42-ff06aa74ac21",
 "created_at": "2016-01-07T18:55:22Z",
 "updated_at": null
},
"entity": {
 "name": "dev",
 ...
```

Work with your containers in IBM Cloud Container Service by using REST. Find all supported IBM Cloud Container Service API calls in the [API documentation](#).

Every API call requires the access token and space ID to be part of the header information as shown in the following table.

*Table 6. Request input parameters to work with the IBM Cloud Container Service API*

| Request input parameter              | Values                                                                                                                                                                             |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IBM Cloud Container Service base URL | https://containers-api.DomainName/v3                                                                                                                                               |
| Header                               | <ul style="list-style-type: none"><li>◦ X-Auth-Token: bearer &lt;access_token&gt;</li><li>◦ X-Auth-Project-Id: &lt;space_ID&gt;</li><li>◦ Content-Type: application/json</li></ul> |

## IBM public, Docker Hub, and private images

A container image is the basis for every container that you create. An image is created from a Dockerfile, which is a file that contains instructions to build the image, and build artifacts, such as an app, the app's configuration, and its dependencies. Think of a container image as an executable file (.exe or .bin). As soon as you run the executable app file, you create an instance of your app. When you run a container, you create a container instance from the image.

Container images are stored in your private Cloud registry and can be added from the CLI only. Every container image that you want to use must be in a registry before you can create a container from it.

When you plan to work with IBM Cloud Container Service, you must decide on the container image that you want to use. There are several ways to create, build and use images in your private Cloud registry.

### **Important**

Each image can have a maximum size of 10 GB to run as single containers or scalable container groups in IBM Cloud Container Service.

### **IBM public images**

Start with one of the [IBM public images](#) that are provided by IBM Cloud Container Service, such as the IBM Liberty and IBM Node images to test the features of IBM Cloud Container Service. Then, you can use one of these images as a parent image, modify the Dockerfile and build your own image with your own app code on it.

### **Images from Docker Hub**

[Copy images directly from Docker Hub](#) into your private Cloud registry or [pull an image from Docker Hub](#), modify it locally, and then build it directly in your registry.

### **Create your own image**

If you have container images that you already use in your local Docker environment, you can [push them to your private Cloud registry](#) to use them in IBM Cloud Container Service. You can also create your own Dockerfile, build, test it locally, and then push it to your private images registry.

### **Viewing the images in your private images registry**

You can view all container images that are available in your private Cloud images registry by using the Cloud GUI or the CLI.

View images that are provided by IBM and images that you previously pushed to your private registry.

From the Cloud GUI, select the catalog and then **Containers**. You can see images that are

provided by IBM and images that you previously pushed to your private Cloud registry.

From the CLI, run `bx ic images`.

## **Reviewing image vulnerabilities**

When you add images to your private Cloud registry, they are automatically scanned by the Vulnerability Advisor against standard policies set by the organization manager and a database of known Ubuntu issues. The Vulnerability Advisor checks on inventory packages, configurations, ports opened, and Docker metadata. When the scan is complete, you can review a list of potential vulnerabilities and address them as necessary before you use the image in a container. Depending on the policies that are set by the organization manager, the deployment of a container from that image can be blocked, or a warning is shown to the user.

## **Dockerfile tips to prevent a container from shutting down**

IBM Cloud Container Service offers a lot of features to keep containers secure and manage cloud resources for the user. Depending on your app, it might be necessary to add extra configurations to your Dockerfile to ensure that your local containers can successfully run in Cloud.

## **Prepare for network delays**

When a container starts, IBM Cloud Container Service sets up the private container network, and assigns a private IP address to the container. This process might take a few seconds. If your app requires an active network connection at the time the app starts, [assure that the IBM Cloud Container Service networking is finished](#) beforehand by adding a sleep command to your Dockerfile.

## **Use long-running commands**

To keep a container up and running at least one long-running process is required to be included in the container image. For example, `echo "Hello world"` is a short running process. If no other command is specified in the image, the container shuts down after the command is executed. To transform the `echo "Hello world"` command into a long running process, you can, for example, loop it multiple times, or include the echo command into another long running process inside your app.

## **Private container network settings in IBM Cloud Container Service**

A container private network creates an isolated and secure environment for the single containers and container groups that run in one space. Containers that are connected to the same private network can send and receive data from other containers in the private network by using the private IP addresses. Containers are not publicly available until a public port and either a public IP address for single containers or a public route for container groups are bound.

You can choose to use the IBM Cloud Container Service default private network settings or to connect your containers to a corporate data center through VPN.

### **IBM Cloud Container Service default private network settings**

In Cloud, every space is already provided with a container private network that applies the default IBM Cloud Container Service network settings. These settings include the automatic set up of a default Private Network Security Group that allows private network communication between containers and container groups by using a private IP addresses. When you create a container or container group in a space, they are automatically connected to the default IBM Cloud Container Service private network and assigned a private IP address from the subnet 172.31.0.0/16. After a private IP address is assigned, all containers in the same network can securely communicate on all container ports by using the private IP address of the single container, or, if using a container group, the private IP address of the group's load balancer. No mapping of container ports to a host port is necessary.

### **Virtual Private Network to connect your containers to a corporate data center**

Securely connect the single containers and container groups in a private container network in Cloud to a corporate data center by using the IBM® Virtual Private Network (VPN) service. IBM VPN provides a secure end-to-end communication channel over the internet that is based on the industry-standard Internet Protocol Security (IPsec) protocol suite. The IPsec protocol offers network-level peer authentication, data integrity, and data confidentiality by encrypting the packages that are exchanged between the VPN endpoints. To set up a secure connection between the containers in Cloud and a corporate data center, you must have an IPsec VPN gateway or SoftLayer server installed in your on-premise data center. With the IBM VPN service, you can configure one VPN gateway per space, and define up to 16 connections to different destinations.

To set up a secure VPN connection to your containers in Cloud, see [IBM VPN](#).

## **Tips to handle network delays when starting a container**

When a container starts, IBM Cloud Container Service sets up the private container network, exposes container ports, and assigns a private IP address to the container. This process might take a few seconds. If your app requires an active network connection at the time the app starts, the container private network might not yet be set up, which can cause the app to crash. To assure that the IBM Cloud Container Service networking is finished before the app starts, consider to implement one of the following solutions to handle network delays.

Choose between the following options to handle network delays.

Add a sleep command to your Dockerfile that is executed when the container starts.

In the following sample Dockerfile, the container waits 60 seconds before the app is started.

```
FROM sdelements/lets-chat:latest
CMD (sleep 60; npm start)
```

Adjust your app code to check for an active network connectivity by, for example, pinging a public IP address before the app starts. If you cannot ping the IP address, sleep for a few seconds and then try again. If the IP address can be resolved, start the app.

## **Determining public network settings**

By default, every single container and container group in IBM Cloud Container Service is available to the private network. However, your app might require accessibility from the internet, or you want to access containers in other spaces of your organization. In these cases, you must expose your container to the public. Depending on the type of container that you choose, different ways exist to make a container available to the public.

### **Single containers and Docker Compose public network settings**

All single containers that you create in your space are assigned a private IP address that you can use to access the containers from the private network only. The private network communication is secured by the default Private Network Security Group that does not allow any network traffic

to and from a public network. If you want to make your app available to the public network, you must expose a public port, and bind a public IP address to your container.

Public IP addresses must be requested for a space to be bound to a container. When requesting a public IP address, a random one from the public IP address pool is allocated to the space. While allocated to a space, the IP address does not change. To bind a public IP address to a container, you must also expose a HTTP port by using the `-p` option in the `bx ic run` command. Exposing a port, creates a Public Network Security Group for your container that allows you to send and receive public data on the exposed port only. All other public ports are closed and cannot be used to access your app from the internet.

**Note**

Binding a public IP address does not include a SSL certificate which encrypts the data that are sent to and from your single container. If your app requires SSL encryption, you can either implement your own SSL solution or use a container group instead of a single container. Container groups are bound to a public route that already includes a SSL certificate, so you can access your container group with HTTPS without any additional configuration.

The number of public IP addresses that are available to a space depends on your organization's [quota](#).

To review the list of available public IP addresses in your space, run the following command.

```
bx ic ips
```

The list that is returned can either show no available public IP addresses, public IP addresses that are bound to a container, or public IP addresses that are unbound.

**No public IP address is available in the space**

CLI output:

| IP Address | Container ID |
|------------|--------------|
|------------|--------------|

My options:

Request a new public IP address and bind it to your container.

```
bx ic ip-request
```

```
bx ic ip-bind <IP_ADDRESS> <CONTAINER>
```

### A public IP address is available in the space and it is not bound to a container

CLI output:

| IP Address | Container ID |
|------------|--------------|
| 192.0.2.56 |              |

My options:

Bind the public IP address to your container.

```
bx ic ip-bind <IP_ADDRESS> <CONTAINER>
```

### A public IP address is available in the space but it is already bound to a container

CLI output:

| IP Address | Container ID                         |
|------------|--------------------------------------|
| 192.0.2.56 | d7b4a167-8e51-4b16-b591-1a17cb7ee41f |

My options:

- Unbind an already bound IP address from a container and bind it to a new container.

```
bx ic ip-unbind <IP_ADDRESS> <CONTAINER>
```

```
bx ic ip-bind <IP_ADDRESS> <CONTAINER>
```

- Request a new public IP address and bind it to your container.

```
bx ic ip-request
```

```
bx ic ip-bind <IP_ADDRESS> <CONTAINER>
```

## Container group public network settings

To make your container group accessible from the internet, you must expose a port and either bind a public route or a public IP address to it.

### Public route with default domain

A public route consists of a host and domain name and composes the full public URL that you enter into your web browser to access the container group. Every route must be unique in Cloud.

You can choose a host name for your route, such as *mycontainerhost* during container creation. The default system domain is AppDomainName and already provides a SSL certificate, so you can access your container group with HTTPS without any additional configuration.

Example: `https://mycontainerhost.AppDomainName`

#### Note

To use a public route, you must expose a HTTP port for your container group during creation. Non-HTTP ports cannot be exposed publicly. You can expose one public port per container group only. Multiple public ports for a container group are not supported.

### Public route with custom domain

If you want to use your own custom domain instead of the default domain AppDomainName, you must register the custom domain on a public DNS server, configure the custom domain in IBM Cloud, and then map the custom domain to the IBM Cloud system domain on the public DNS server. After your custom domain is mapped to the IBM Cloud system domain, requests for your custom domain are routed to your application in IBM Cloud. When you create a custom domain, do not include underscores (\_) in the domain name.

To create a custom domains, see [Creating and using a custom domain](#).

To make your custom domain secure, [upload a SSL certificate](#), so your container groups can be accessed with HTTPS.

### Public IP address



You can also bind an IP address when creating a container group with the command line. You can only bind a public route or a floating IP address to a container, not both. To remove an IP address from a container group, you must remove the container group and create the group again.

Example of creating a container group with a public IP address:

```
bx ic group-create -p 9080 --ip 192.0.2.56 --name my_container_grou
```

## Integrating Cloud services to use with containers

Cloud offers various services that you can use with IBM Cloud Container Service to bring extra capabilities to your app or to simplify the implementation of your app and the management of your container.

You can see all the services that are available in Cloud in the following ways:

- From the Cloud user interface, select the Cloud Catalog.

- From the CLI, run the `cf marketplace` command.

To use a Cloud service with IBM Cloud Container Service, you must [create an instance of this service](#) in your space before you can create a container. After a service instance is created in your space, you must [bind the service to your container](#) so that the container can access the environment variables that define the service. These environment variables are called `VCAP_SERVICES`. When you bind a service to a container, the service injects the `VCAP_SERVICES` as environment variables in the container. In this way, the container can use the `VCAP_SERVICES` to look up information from the service, such as the endpoint, the user name, and password.

To find information about services that help you manage your containers, see [Integrating services with single and scalable containers \(Deprecated\)](#).

## Persistent data storage options

Decide where to store the data that the app creates or the files that the app requires to run.

A single container is, by design, short-lived. However, there are several options that you can choose from to persist data between container restarts, to share data between containers in a

space, and to share data between container instances in a group.

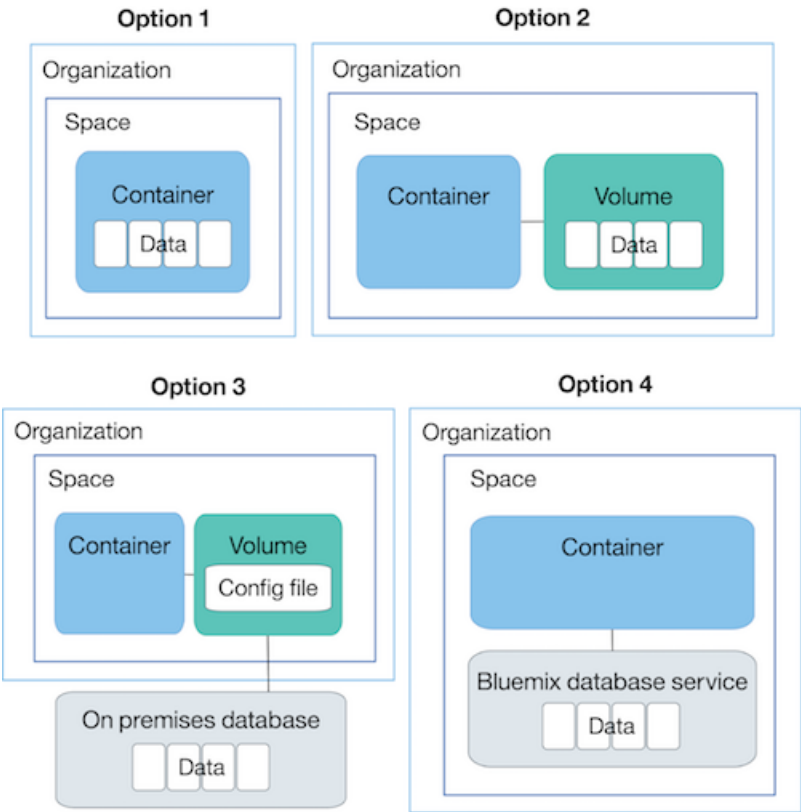


Table 7. Options for storing app files

| Data storage options                                       | Description                                                                                                                                                                                                 | Amount of data that can be stored is limited by |
|------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| Option 1: Store the data inside a container during runtime | With this option, you do not persist app data. Data are available to the container during runtime only. When a container is re-created, all the data is lost.                                               | Size of the container                           |
| Option 2: Store the data in a container volume             | When you mount a volume in Docker, the volume is mounted to your local file system. In IBM Cloud Container Service, the access to the compute host is restricted, so you cannot mount host directories to a | Size of the file share that hosts the volume    |

| Data storage options | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Amount of data that can be stored is limited by |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
|                      | <p>container. Instead, organization-scoped volumes are used to persist data between container restarts. Volumes are hosted on isolated file shares that securely store app data and manage the access and permission to the files. Due to this isolation, non-root users inside the container must be <a href="#">granted write permission to the mounted volume</a>.</p> <p>With this option, you can persist and access data between container restarts, and share data between containers in a space. When a container is deleted, the associated volume is not removed.</p> <div data-bbox="594 1409 1029 1862"><p><b>Note</b></p><p>To mount a volume to a container, you must create one first, see <a href="#">Create a volume with the Cloud GUI</a>.</p></div> |                                                 |

| <b>Data storage options</b>                                | <b>Description</b>                                                                                                                                                                                                                                | <b>Amount of data that can be stored is limited by</b>  |
|------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| Option 3: Connect your container to an on-premise database | With this option, you can persist and access data in an existing on-premise database by using a <a href="#">secured VPN connection</a> between your containers in the Cloud and your local environment.                                           | Disk space that is available to the on-premise database |
| Option 4: Bind a Cloud database service to your container. | With this option, you can persist and access data by using a database service that is linked to your container in the Cloud. Most Cloud database services provide disk space for a small amount of data at no cost, so you can test its features. | Database service and plan that you choose               |

## **Non-root user access to container volumes in Cloud**

If you decide to use organization-scoped container volumes to persist app data, ensure that the user inside the container has write access to files and folders on the volume mount path. For IBM Cloud Container Service, the user namespace feature is enabled for Docker Engine. User namespaces provide isolation so that the container root user cannot gain access to other containers or the compute host. However, volumes are on NFS file shares that are external to the container compute hosts. The volumes are set up to recognize the root user in the container, but because volumes are external to the container they are not aware of user namespaces.

For some applications, the only user inside a container is the root user. However, many applications specify a non-root user that writes to the container mount path. If you are designing an application with a non-root user that requires write permission to the volume, you must add the following processes to your Dockerfile and entrypoint script:

Create a non-root user.

Temporarily add the user to the root group.

Create a directory in the volume mount path with the correct user permissions.

For more information, see [create your own Dockerfile with the appropriate non-root permissions](#).

## Container types

Before you create a container with IBM Cloud Container Service, decide on the type of container that you need. IBM Cloud Container Service offers different approaches to create containers in the cloud. The approach that you choose depends on the requirements and dependencies of your app that runs in your container.

### Single container

A single container in IBM Cloud Container Service is similar to a container that you create in your local Docker environment. Single containers are a good way to start with IBM Cloud Container Service and to learn about how containers work in the IBM cloud and the features that IBM Cloud Container Service provides. You can also use single containers to run simple app tests or during the development process of an app. Because a single container can be restarted and is not intended for hosting a long-running program, you must use a container group for any application that requires high availability.

Deployment speed considerations for single containers:

The size of the image has a significant impact. The smaller the image, the quicker the deployment.

After the first few times an image is deployed, deployment speeds improve. Initially, the image must be downloaded to the registry on the host. Subsequent deployments are faster.

The networking setup might take a few minutes.

A single container deploys faster than a container group because of the routing setup for groups.

Deployments with linked containers might not be as quick as other deployments because of the connections that must be made.

To create a single container from the Cloud GUI, see [Running short-term tasks as single containers \(Deprecated\)](#).

To create a single container from the CLI, see [Running short-term tasks as single containers with the command line interface \(CLI\)](#).

## Container groups

A container group consists of multiple single containers that are all created from the same container image and as a consequence are configured in the same way.

The number of container group instances that your app requires depends on the expected workload and the time in which a request can be processed by the app. For example, multi-tasking or network connectivity affects how many instances might be required for your app. To determine the number of desired instances, deploy a single container in Cloud that runs your app and perform a load test on this container. If the app can handle, for example, 100 requests per second, but you are expecting an average workload of 300 requests per second, then the desired number of instances for the container group is 3. To make your container group highly available and more resilient to failure, consider including extra instances than the minimum to handle the expected workload. Extra instances can handle the workload in case another instance crashes. To ensure automatic recovery of a failed instance without affecting the workload, include one extra instance. For protection against two simultaneous failures, include two extra instances. This set up is an  $N+2$  pattern, where  $N$  is the number of instances to handle the requests and  $+2$  is an extra two instances.

A container group offers further options at no cost to make your app highly available.

[Built-in load balancing](#)

[Anti-affinity](#) to spread container group instances across multiple physical compute nodes

[Auto-recovery](#) of unhealthy container group instances

[Auto-scaling](#) of container group instances based on CPU and memory usage

To increase the availability of a container group even more, you can create another container group that is based on the same image and is mapped to the same route. When the first container group crashes, the second one can take over the entire workload. Choose between the following options to spread the second container group across the IBM Cloud Container Service architecture.

Create a second container group within the same space.

Spread container groups across multiple Cloud regions.

Deployment speed considerations for container groups:

The size of the image has a significant impact. The smaller the image, the quicker the deployment.

After the first few times an image is deployed, deployment speeds improve. Initially, the image must be downloaded to the registry on the host. Subsequent deployments are faster.

The networking setup might take a few minutes.

A single container deploys faster than a container group because of the routing setup.

A container group that is not bound to a route deploys quicker than one without a route.

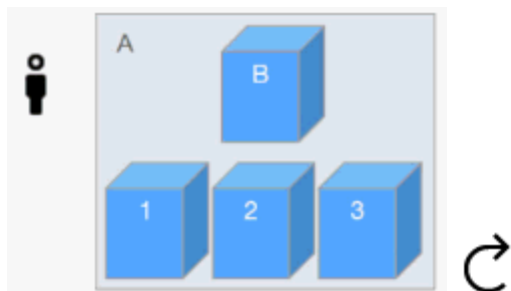
Container groups that do not have anti-affinity enabled are faster than container groups with anti-affinity. With anti-affinity, the number of hosts that is specified must each download the image layers.

To create a container group from the Cloud GUI, review the [Run a container group from the Cloud GUI](#) topic.

To create a container group from the CLI, see [Run a container group from the CLI](#).

### Load balancing for a container group

Container groups provide in-built load balancing that routes incoming traffic evenly (round-robin) to the container instances in the group. When a container group is created or instances are added to an existing group, whether you add instances yourself or an instance is re-created during auto-recovery, load balancing is enabled.



**A**: Container group with three instances

**B**: Load balancer for the container group

## **1, 2, 3**: Instances in the container group

After a new instance is added to the container group, the load balancer makes a TCP socket connection to determine when to start sending traffic to the new container instance. As soon as the container instance starts listening on its defined port, the load balancer starts sending traffic to that instance. The load balancer assigns the group a private IP address so that the group can be accessible to the other containers or services in the same space.

Because this private IP address is assigned to the group and not to individual instances, the private IP address does not change when you add instances to the group or an instance is re-created during auto-recovery. You can find this private IP address by running `bx ic group-inspect`.

### **Anti-affinity for a container group**

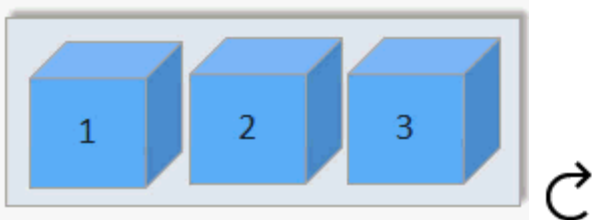
You can choose whether to enable the anti-affinity feature when you create a container group by using the `--anti` option in the `bx ic group-create` command.

When anti-affinity is enabled, the container instances are spread across separate physical compute nodes, which reduces the likelihood of containers crashing due to hardware failures.

To use the `--anti` option from the CLI, you must have installed the IBM Cloud Container Service plug-in (`bx ic`) version 0.8.934 or later. You might not be able to use this option with larger group sizes because each Cloud region and organization has a limited set of compute nodes available for deployment. If your deployment does not succeed, either reduce the number of container instances in the group or remove the `--anti` option.

### **Auto-recovery for a container group**

You can choose whether to enable auto-recovery when you create a container group by using the `--auto` option in the `bx ic group-create` command.



When automatic recovery is enabled for a container group and the group has been running for



10 minutes, the group's load balancer starts to regularly check the health of each container instance in the group via HTTP requests. If a container instance does not respond within 100 seconds, it is marked as inactive. Inactive container instances are removed from the group and re-created by auto-recovery. Auto-recovery attempts to recover container instances in a group 3 times. After the third attempt, auto-recovery does not recover any container instances in the group for 60 minutes. After 60 minutes, the auto-recovery process starts again.

### **Auto-scaling for a container group**

Container groups offer the possibility to auto-scale container group instances based on the CPU and memory that is used by the app.

You can define auto-scale policies that determine when a container instance is added to, or removed from the group. In this way, you can automatically scale your container group based on your app's workload. To learn more about auto-scaling of container group instances, see [Automatically scaling container groups \(deprecated\)](#).

### **Docker Compose**

Some apps consist of multiple components, such as database, services and caches, where each component must run in its own container. As orchestrating all containers to start up, link and shut down together can be very difficult, you can use Docker Compose to configure your multi-container deployment and to run it with one command only. Think of Docker Compose as a collection of single container instances where every container is based on its own image and configuration. You configure your multi-container deployment one time and deploy it in Cloud.

#### **Note**

Docker Compose does not support the IBM Cloud Container Service group concept, which is why you must use your own tools to address load balancing, auto-recovery and auto-scaling for your app.

Deployment speed considerations for Docker Compose:

A Docker Compose deployments might not be as quick as other deployments because coordinated deployments are being set-up.

The size of the image has a significant impact. The smaller the image, the quicker the deployment.

After the first few times an image is deployed, deployment speeds improve. Initially, the image must be downloaded to the registry on the host. Subsequent deployments are faster.

The networking setup might take a few minutes.

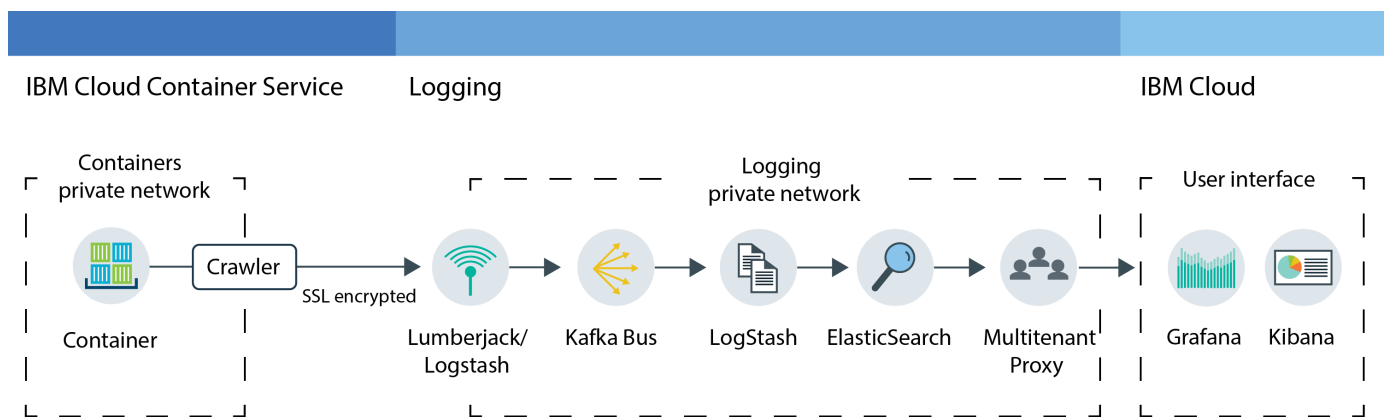
## Monitoring and Logging

In Cloud, containers come with built-in monitoring and logging, so you can watch the health of your containers closely.

### Important

This service is not intended for processing sensitive data.

IBM Cloud offers platform-based monitoring and logging capabilities. Metric and log data can be collected from compute instances, like containers, that are running in the cloud and from other sources. Aggregated data from multiple sources can then be visualized for analysis, insights, and action.



## Monitoring

Container metrics are collected from outside of the container, without having to install

and maintain agents inside of the container. In-container agents can have significant overheads and setup times for short-lived, lightweight cloud instances and auto-scaling groups, where containers can be rapidly created and destroyed. This out-of-band data collection approach eliminates these challenges and removes the burden of monitoring from the users.

A process that is running in the host performs agentless monitoring for metrics. This process, which is also called the *crawler*, constantly collects the following information from all of the containers.

- CPU

- Memory

- Network information

This metric data is collected in Graphite and is displayed in both the Cloud user interface and the Advanced monitoring view. Docker conventions and groups accounting information are used as the basic mechanism for the collection of monitoring data.

You can see a list of plug-ins that are installed to collect the data in `/etc/collectd/collectd.conf`. To learn more about what those plug-ins do, see the [collectd documentation](#).

## Logging

Similar to metrics, container logs are monitored and forwarded from outside of the container by using crawlers. The data is sent by the crawlers to a multi-tenant [Elasticsearch](#) in IBM Cloud, just like logs that are collected by other in-container agents, but without the hassle of having to install the agents inside the container. The crawlers retrieve information from these locations by default.

- `/var/log/messages`: System messages.

- `/docker.log`: The Docker logs.

The Docker log file is not stored as a file inside of the container, but it is collected anyway. This log file is collected by default as it is the standard Docker convention for exposing the `stdout` (standard output) and `stderr` (standard error)

information for the container. If any container process prints to `stdout` or `stderr`, that information is collected.

To collect additional logs, add the `LOG_LOCATIONS` environment variable with a path to the log file when creating your container in the [GUI](#) or the [CLI](#).

To see more on how to collect logs and metrics, as well as how to create custom dashboards, see [Monitoring and logging containers](#)

# Security for single and scalable containers (Deprecated)

IBM® Cloud Container Service provides a trusted and secured cloud platform that optimizes Linux and Docker security capabilities. When you build and run containers with IBM Cloud Container Service, your app benefits from container and network isolation, limited access to host data and resources, and secure container deployments.

## Attention

Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

In this page:

- [Built-in security settings](#)
- [Compute host security settings](#)
- [Private container network security settings](#)
  - [Logging into a container using exec](#)
  - [Connecting containers in Cloud to a corporate data center via VPN](#)

## Built-in security settings

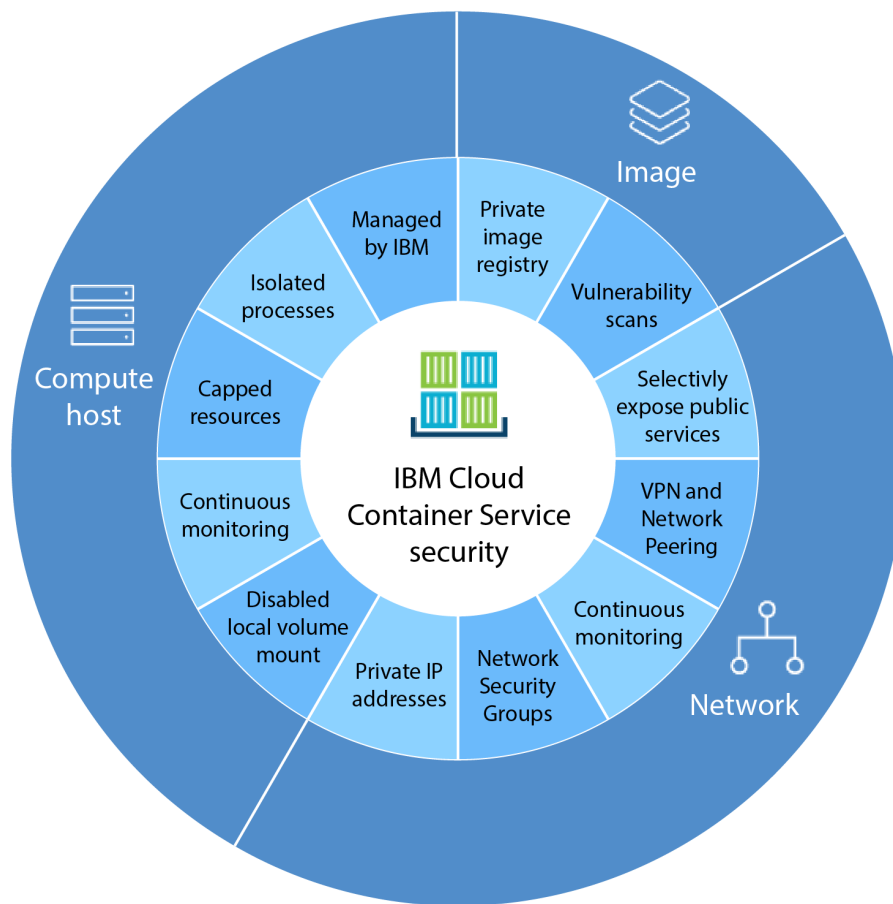


Table 1. Built-in IBM Cloud Container Service security settings

| Level                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compute host              | <p>All containers run as independent and isolated processes on the compute hosts and have restricted access to its resources. Without additional configuration, built-in security settings help to protect the Linux kernel and Docker daemon from Denial-of-Service (DOS) attacks.</p> <p>For more information, see <a href="#">Compute host security</a>.</p>                                                                                                  |
| Private container network | <p>Every container runs in an isolated and secured private network where inbound and outbound network traffic is monitored to detect and remediate malicious activity. You can set up your app without public network access or selectively expose services to the public.</p> <p>To securely connect containers to a corporate data center or to other Cloud spaces, <a href="#">you can set up a Virtual Private Network by using the IBM VPN service</a>.</p> |

| Level            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Docker image     | <p>Every organization is assigned a secured Docker v2 private image registry where images are stored and shared by all users in the organization.</p> <p>When logging in to the IBM Cloud Container Service for the first time, every organization is required to set a namespace. The namespace is used to create a unique URL that identifies the organization's private registry in the following format: <code>registry.DomainName/namespace</code>.</p> <p>To add images to your private registry, you can create your own image, add an existing local image, or copy an image directly from Docker Hub.</p> <p>To ensure safe container deployments, every image is scanned by Vulnerability Advisor. Vulnerability Advisor scans for potential vulnerabilities, makes recommendations, and provides instructions to resolve vulnerabilities.</p> |
| Docker container | <p>Vulnerability Advisor checks the status of running containers in your organization to ensure containers continue to be secure and compliant with an organization's policy. Vulnerability Advisor scans for potential vulnerabilities, makes recommendations, and provides instructions to resolve vulnerabilities.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

## Security settings on the IBM Cloud Container Service compute host

IBM Cloud Container Service optimizes the Linux kernel and Docker security capabilities to ensure container isolation and to restrict access to compute host resources.

### Fully managed and secured Docker daemons

The Docker daemons that run on the IBM compute hosts are set up without direct user access and can be configured only by IBM. All Docker daemon sockets are secured with TLS certificates.

### Enabled Docker user namespaces

The [Docker user namespace](#) feature is enabled in the Docker Engine on the compute hosts. The user namespace feature isolates processes that run inside a container from all other processes and containers that run on the same compute host. The effective root user inside the container is mapped to a non-root user on the compute

host. Even if a root user managed to break out of the container, the user does not have any permissions to manipulate files, processes, or other containers that run on the compute host.

### **No privileged containers allowed**

The `--privileged` option in the `run` command is not supported. Containers do not have access to devices and hard disks on the compute host.

### **Managed control groups (cgroups) for memory, CPU, and disk space**

Containers that run on the same compute host share memory, CPU, and disk space.

To prevent a container from either consuming all available resources or bringing down applications or host systems, the maximum amount of memory, CPU, and disk space that can be assigned to a container is limited.

The amount of memory, CPU, and disk space that is available to the container during runtime is determined by the container size that you select during container creation.

This limitation ensures that every container on the compute host gets its fair share of the available host resources. If a container misbehaves, the resource capping ensures a consistent performance and behavior for other containers on the compute host.

### **Compute host kernel optimization**

Compute host kernels limit the total number of threads and processes that can run on the compute host by a specific user or group. This optimization ensures that the host is not overloaded and does not run out of process slots until the user's maximum is reached.

### **Continuous compute host monitoring**

Every compute host is continuously monitored by IBM to control and remediate fork-bombs and other process level Denial-Of-Service (DOS) attacks.

### **Enabled AppArmor profile for Docker containers**

Every compute host applies the IBM-specific [AppArmor](#) profile to secure the environment and restrict the capabilities of a container on the compute host.

AppArmor is a Linux kernel security module that controls the access to folders, files, network domains, and permissions to create and change data.

### **Disabled local volume mount on compute hosts**

IBM Cloud Container Service does not support the mounting of local volumes or `local/host` directories to a container. Instead, organization-scoped volumes are used to persist data between container restarts and to make data available to multiple containers. Volumes are hosted on isolated file shares that securely store app data and manage the access and permission to the files.



Organization-scoped volumes must be created before you can mount them to your container. You can create a volume from the [Cloud GUI](#) or the [CLI](#).

## Security settings on the IBM Cloud Container Service private container network

Review the list of integrated IBM Cloud Container Service features that ensure secure communication between containers in the same private network, between containers in different Cloud spaces, and between your Cloud space and a remote network.

### **Private L2 isolated container network per space**

A container private network creates an isolated and secure environment for the single containers and container groups that run in one space. Containers that are connected to the same private network can send and receive data from other containers in the private network by using the private IP addresses. Containers are not publicly available until a public port and either a public IP address for single containers or a public route for container groups are bound.

In Cloud, every space is provided with a container private network where the default IBM Cloud Container Service network settings are applied. When you create containers, they are automatically connected to the default private network for the space and assigned a private IP address.

### **Keep containers private or selectively expose public facing services through Network Security Groups**

Every container is protected by IBM-specific Network Security Groups that control the inbound network traffic and serve as a software firewall on the private network gateway level. Data that is sent through the private network gateway is checked for compliance against the Network Security Group rules and can either be passed to the recipient container or blocked.

When you create a container and do not expose it publicly, it is set up with the default Private Network Security Group. This security group allows private network communication between containers by using the private IP addresses. By default, all container ports are automatically exposed on the private network. You do not have to map the container ports to a host port.

To make your app available publicly, you must expose a public port and bind either a public IP address to your single container, or a public route to your container group. When you expose a public port by using the `-p` option in the `bx ic run` command, you create a Public Network Security Group for your container that allows public data to be sent and received on the exposed port only. All other public ports are closed and cannot be used to access your app from the internet.

### **Continuous enterprise network monitoring**

All network traffic in IBM Cloud Container Service is monitored by IBM to detect and remediate malicious activities.

### **Full support for the Docker `exec`, `attach`, and `logs` commands**

You can log in to your container or view its detailed output during runtime by leveraging the `bx ic exec`, `bx ic attach`, or `bx ic logs` commands. With these commands, you can avoid running a non-secure SSH daemon to access and monitor the container during runtime.

For more information about how to securely log in to your container by using `bx ic exec`, see [Logging in to a container with `exec`](#).

### **VPN for secure private network access to services in other spaces and in any remote network**

Containers in a private container network can be securely connected to corporate data centers by using the IBM® Virtual Private Network service.

For more information, see [Setting up a VPN connection between your containers and a corporate data center](#)

## **Logging in to a container with `exec`**

If you must log in to your running container, you can use `bx ic exec`.

Common uses of the `bx ic exec` command include opening a bash window and viewing files of a volume assigned to a container. The `bx ic exec` command only runs on a container that is in a running state. To keep a container up and running, the container must be started with a long-running command. If a container is paused or stopped, it must be restarted with a long-running command before you can log into it.

### **Command and parameters**

```
bx ic exec [-d] [-it] [-u USER] CONTAINER CMD
```

### Note

\*In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands.

## Examples

This example command runs an interactive bash terminal in the container *my\_container*.

```
bx ic exec -it my_container bash
```

This example command runs the `date` command in the container *my\_container*.

```
bx ic exec my_container date
```

## Connecting containers to a corporate data center by setting up a Virtual Private Network

Securely connect the single containers and container groups in a private container network in Cloud to a corporate data center by using the IBM® Virtual Private Network (VPN) service. IBM VPN provides a secure end-to-end communication channel over the internet that is based on the industry-standard Internet Protocol Security (IPsec) protocol suite. The IPsec protocol offers network-level peer authentication, data integrity, and data confidentiality by encrypting the packages that are exchanged between the VPN endpoints. To set up a secure connection between the containers in Cloud and a corporate data center, you must have an IPsec VPN gateway or SoftLayer server installed in your on-premise data center. With the IBM VPN service, you can configure one VPN gateway per space, and define up to 16 connections to different destinations.

To connect containers to a corporate data center, follow these steps.

Create at least one [single container](#) or [container group](#) in your space that you want to connect to your corporate data center.

Create an instance of the IBM VPN service in your space by following step 1-5 of the [Binding a service](#) topic.

**Note:**

The IBM VPN service instance must be created in the same space where you created the containers that you want to connect to the corporate data center.

Follow the instructions in the [IBM VPN getting started](#) topic to configure the VPN gateway and set up connections to your corporate data center.

# Setting up the IBM Cloud Container Service plug-in (bx ic) to use the native Docker CLI with single and scalable containers (Deprecated)

Install the IBM® Cloud Container Service plug-in (bx ic) to run native Docker CLI commands to manage your containers.

## Attention

Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

In this page:

- [Installing the IBM Cloud Container Service plug-in \(bx ic\)](#)
- [Logging in to the IBM Cloud Container Service CLI plug-in \(bx ic\)](#)
- [Upgrading the IBM Cloud Container Service plug-in \(bx ic\)](#)
- [Uninstalling the IBM Cloud Container Service plug-in \(bx ic\)](#)
- [Installing Docker Compose and its dependencies](#)
- [Upgrading Docker Compose](#)
- [Uninstalling Docker Compose](#)

## Installing the IBM Cloud Container Service plug-in (bx ic)

To run commands, your operating system must meet these requirements. This task includes the information for installing these requirements. Earlier versions of these tools are not supported for use with the IBM Cloud Container Service plug-in (bx ic).

- Docker version 1.8 to 1.12
- Cloud Foundry CLI version 6.14.0 - 6.22.0

## Note

MacOS Sierra users must install Cloud Foundry CLI version 6.22.0 or later.

Cloud CLI

IBM Cloud Container Service plug-in

To install the CLIs:

Install Docker 1.12. For Windows and OS X operating systems, Docker is installed along with the Docker Toolbox, which includes the Docker Machine, Docker Engine, and Kitematic. For Linux operating systems, installation mechanisms vary depending on the type of Linux distribution you use. Refer to the following links to install or, if you already installed Docker, to upgrade to Docker 1.12.

### Note:

If you already installed Docker on your computer and you do not want to upgrade your current version, you can skip this step and continue with installing the Cloud Foundry CLI.

- o OS X

<https://github.com/docker/toolbox/releases/download/v1.12/DockerToolbox-1.12.pkg>

- o Windows

<https://github.com/docker/toolbox/releases/download/v1.12/DockerToolbox-1.12.exe>

- o Linux

Refer to the [Docker installation documentation](#) to find instructions on how to install Docker version 1.12 on the Linux distribution you use.

Start Docker.

- o For OS X and Windows, double-click the **Docker Quickstart Terminal** icon. The Docker daemon is started and environment variables are set for you.
- o For Linux, in a command line utility, verify that the Docker daemon is running. Also, set up the Docker daemon to restart automatically. For more information, [see the Docker documentation for your operating system](#).

Verify that your Docker installation is set up correctly by running `docker run hello-world` in your terminal.

```
docker run hello-world
```

Docker looks for the `hello-world` image locally to run a container from, so if this is your first time running the command, you receive the following message.

```
Unable to find image 'hello-world:latest' locally
```

The image is pulled from Docker Hub, a public repository of images, and after the command completes, your output looks like the following:

```
$ docker run hello-world
```

```
Hello from Docker.
```

This message shows that your installation appears to be working correctly

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker Hub account  
<https://hub.docker.com>

For more examples and ideas, visit:

```
https://docs.docker.com/userguide/
```

If the command is not completed properly, run `docker version` to see if Docker is installed correctly.

Install Cloud Foundry CLI version Cloud Foundry CLI version 6.14.0 - 6.22.0 from the [GitHub repository](#). You must install the Cloud Foundry CLI in the default location for your operating

system, otherwise the PATH environment variable does not match your installation directory.

**Note:**

MacOS Sierra users must install Cloud Foundry CLI version 6.22.0 or later.

To verify that the Cloud Foundry CLI is installed properly or, if you already installed the Cloud Foundry CLI, to check which version you installed, run the following command:

```
cf -v
```

Install the [Cloud CLI](#).

Log in to the Cloud CLI.

```
bx login -a https://api.ng.bluemix.net
```

Install the IBM Cloud Container Service plug-in.

```
bx plugin install IBM-Containers -r Bluemix
```

To verify that the plug-in is installed properly, run the following command:

```
bx plugin list
```

The plug-in is displayed in the results as IBM-Containers.

You are ready to log in and begin managing containers.

## Logging in to the IBM Cloud Container Service CLI plug-in (bx ic)

After you install the CLI, log in to use it.

Before you begin, complete the following tasks.

[Install the IBM Cloud Container Service \(bx ic\) plug-in.](#)

Decide which Cloud organization and space to use when you log in to Cloud Foundry. For more information about organizations and spaces, see [Orgs and spaces](#).

For more information about the options you can specify with the Cloud Foundry login command, see [the Login command](#) in the Cloud Foundry documentation.



Open a command line utility.

- For OS X and Windows, double-click the **Docker Quickstart Terminal** icon. The Docker daemon is started and environment variables are set for you.
- For Linux, open any command line utility.

Log in to Cloud through the Cloud Foundry CLI. To specify a specific region, include the `-a` option with the API endpoint for the region. If it is your first time you are using the CLI and you do not include the option, you are prompted to enter an API endpoint. After the API endpoint is specified, you do not need to enter one during subsequent logins.

```
bx login [-a api.DomainName] [-sso]
```

To log in to a specific region, include the API endpoint for that region when you log in to the Cloud Foundry CLI.

- US South

```
bx login -a api.ng.bluemix.net
```

- United Kingdom

```
bx login -a api.eu-gb.bluemix.net
```

The single-sign-on parameter `--sso` is required when you log in with a federated ID. If this option is used, open <https://login.ng.bluemix.net/UAALoginServerWAR/passcode> to obtain a one-time passcode. If you do not include the `--sso` option, complete these substeps.

For **Email**, enter your IBMId for Cloud.

For **Password**, enter the password for the IBMId. Your Cloud organizations and spaces are retrieved.

Enter the number that represents one of your Cloud organizations.

Enter the number that represents one of your existing Cloud spaces.

Set a *namespace* for your organization. A namespace is a unique name to identify your private Cloud images registry. When you create a container, you must specify an image's location by including the namespace with the image name.

#### Note

When you create a namespace, it cannot be changed after it is created, only deleted. If users in your Cloud organization already deployed containers and set a *namespace*, a message is shown displaying the namespace for your organization. In this case, you can continue with the next step.

```
bx ic namespace-set <my_namespace>
```

### Tip

Observe the following rules for the namespace:

- It can contain only lowercase letters, numbers, or underscores (\_).
- It can be 4 - 30 characters. If you plan to manage containers from the command line, you might prefer to have a short namespace that can be typed quickly.
- Its name must be unique in Cloud.

Initialize IBM Cloud Container Service.

```
bx ic init
```

Decide if you want to run `bx ic` or native Docker commands. You can select one of these options every time you log in to the plug-in or you can use the `shellinit` command to always select Option 2 for you.

- **Option 1** (Default)

A simple shell for IBM Cloud Container Service wrappers Docker. The local Docker daemon is unaffected by IBM Cloud Container Service. `bx ic` commands are used to manage containers in your Cloud space and `docker` commands are used to manage containers locally. `docker` commands are always required for local image development though.

For example, with Option 1, the `bx ic run` command creates a container in your Cloud

space and `docker run` creates a container locally. You might prefer this option if you are new to both Docker and IBM Cloud Container Service or if you like to test containers locally.

#### ○ **Option 2**

Redirect your local Docker daemon for the current CLI session to IBM Cloud Container Service so that `docker` commands can be used in place of `bx ic` commands whenever they are available to manage containers in your Cloud space.

For example, with Option 2, either the `bx ic run` or the `docker run` command can be used to create a container in your space. You might prefer this option if you are an experienced Docker user who is more comfortable with the `docker` commands or if you are adapting automated scripts to work with IBM Cloud Container Service.

Some exceptions still require the use of `bx ic` commands to perform tasks specific to IBM Cloud Container Service. For example, to set a namespace, you must use the command `bx ic namespace-set`. `docker namespace-set` is not a valid command. For a list of `docker` commands that are supported with this configuration, see [Supported Docker commands for IBM Cloud Container Service plug-in \(bx ic\)](#).

To configure this option, copy and paste the environment variables that are provided in the terminal during log in.

- `DOCKER_HOST` sets the Docker host to IBM Cloud Container Service.
- `DOCKER_CERT_PATH` sets the location of the authentication keys for the IBM Cloud Container Service plug in for Cloud Foundry.
- `DOCKER_TLS_VERIFY` uses TLS and verifies the remote server.

#### **Tip**

When you are completing the tasks in the IBM Cloud Container Service documentation, you can run `docker` commands in all steps that are marked with an asterisk (\*).

# Upgrading the IBM Cloud Container Service plug-in (bx ic)

You might want to update the plug-in periodically to use new features.

Table 1. Current version of the IBM Cloud Container Service plug-in

| Release date      | Version | Updates                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| November 16, 2016 | 0.8.964 | <ul style="list-style-type: none"><li>Added support for managing the health check for container groups through the <code>bx ic group-create</code> command by using the <code>--http_monitor_enabled</code>, <code>--http_monitor_path</code>, and <code>--http_monitor_rs_list</code> options</li><li>Added support for the <code>bx ic group-create --session_affinity</code> option</li><li>Defect fixes</li></ul> |

To update the plug-in, follow these steps.

Install the update from the plug-in repository that you configured when you first installed the plug-in.

```
bx plugin update IBM-Containers -r Bluemix
```

Verify the plug-in installation by running the following command and checking the list of the plug-ins that are installed.

```
bx plugin list
```

The plug-in is displayed in the results as IBM-Containers.

## Uninstalling the IBM Cloud Container Service plug-in (bx ic)

If you no longer need the IBM Cloud Container Service plug-in, you can uninstall it with a `bx` command.

## Note

To uninstall the IBM Cloud Container Service Cloud Foundry plug-in, run the following command.

```
cf uninstall-plugin IBM-Containers
```

To uninstall the IBM Cloud Container Service plug-in for the Cloud CLI, run the following command.

```
bx plugin uninstall IBM-Containers
```

Verify the plug-in was uninstalled by running the following command and checking the list of the plug-ins that are installed.

```
bx plugin list
```

The IBM Cloud Container Service plug-in is not displayed in the results.

## Installing Docker Compose and its dependencies

To start and stop multiple single containers at one time, you can install the Docker Compose CLI.

To run Docker Compose commands with IBM Cloud Container Service, you must have the following software installed.

- Docker version 1.10.0

- Cloud Foundry CLI version 6.14.0 - 6.22.0

- IBM Cloud CLI

- IBM Cloud Container Service plug-in

- Docker Compose version 1.6.0 or later

This task includes the information for installing these requirements. Earlier versions of these tools are not supported for use with the IBM Cloud Container Service plug-in (bx ic).

[Install Docker, the Cloud Foundry CLI, the IBM Cloud CLI, and the IBM Cloud Container](#)

[Service plug-in.](#)

**Note:**

If you use a Mac or Windows operating system, Docker Compose is automatically installed with the Docker Engine, Machine, and Kitematic as part of the Docker Toolbox installation. You can skip the next step and continue to verify your Docker Compose version.

If you are using Linux, install the latest version of Docker Compose.

Install pip. The package manager pip includes all tools to install, upgrade, and remove software packages that are written in Python on your computer. You can use the following command to install pip or you can download the software package directly to your computer from the [Python Package Index](#).

**Note:**

If you already installed pip on your computer, you can continue with installing the latest version of Docker Compose.

```
easy_install pip
```

Install Docker Compose

```
pip install docker-compose==1.6.2
```

Verify the Docker Compose version that is installed on your computer.

[Log in to the IBM Cloud Container Service CLI](#). Set the environment variables for **Option 2** when you log in to the plug-in to run native Docker commands.

Verify the version of your Docker Compose package.

```
docker-compose version
```

If `docker-compose version 1.6.0` or later is displayed, start creating your own multi-container app. If not, [upgrade your Docker Compose version](#).

**Note:**

If you experience problems with executing Docker Compose commands, review the [No module named queue](#) and [Insecure HTTPS connection warning](#) topic to check whether you need to change system settings or install additional software packages.

## Upgrading Docker Compose

You might need to upgrade the version of your Docker Compose package to use the latest features.

Upgrade Docker Compose.

```
pip install docker-compose --upgrade
```

Verify the version of your Docker Compose package.

```
docker-compose version
```

## Uninstalling Docker Compose

If you no longer need the Docker Compose software package, you can uninstall it with the following command.

Uninstall Docker Compose.

```
pip uninstall docker-compose
```

Verify the Docker Compose package was uninstalled. If Docker Compose was uninstalled successfully, it is not listed in the CLI output when you run the following command.

```
pip freeze
```

# Running short-term tasks as single containers (Deprecated)

Use a single container to run simple tests as you develop an app or service or for short-term processes that do not require high availability. To use your own networking front end, you can assign a public IP address to the container.

## Attention

Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

In this page:

- Running short-term tasks as single containers from the [GUI](#) or the [CLI](#)
- [Updating containers](#)
- [Linking containers](#)
- [Requesting and binding IP addresses](#)
- [Removing single containers](#)

## Running short-term tasks as single containers from the Cloud GUI

Create and deploy containers from the Cloud GUI.

Before you begin, consider the following information.

- You must have a Developer role for the Cloud space to create a container in it. For more information on the permissions of each roles and how to edit roles, see [Users and roles](#).
- From the Cloud GUI, you create a container from either the IBM-provided public images, or from an existing image in your organizations private Cloud registry. For more information



about adding an image to your Catalog, see [Adding Docker images to your organization's private Cloud images registry \(Deprecated\)](#).

Check to see whether an IP address for the space exists that you can use instead of creating a new one. For more information, see [Finding existing public IP addresses for your organization with the command line interface \(CLI\)](#).

If you know that you want to bind a Cloud service to the container, add the Cloud service to your space. For more information, see [Binding a service from the Cloud GUI](#).

If you know that you want a shared disk that allows files to be accessible from within a container, create a volume before you create the container. For more information, see [Creating volumes with the command line \(CLI\)](#).

To create and deploy a container with an image:

From the catalog, select **Containers** and choose an image. Both images that are provided by IBM and images that are stored in your private Cloud registry are displayed.

If a namespace is already set for your organization, you are not prompted to create one and can continue with the next step. If a namespace is not specified for your organization yet, you are prompted to set one for the image registry. A namespace is a unique name to identify your private registry in Cloud. The namespace is assigned one time for an organization and cannot be changed after it is created. If you want to use IBM Cloud Container Service in multiple Cloud regions, you must set up a namespace for each region.

**Tip:**

Observe the following rules for the namespace:

- It can contain only lowercase letters, numbers, or underscores (\_).
- It can be 4 - 30 characters. If you plan to manage containers from the command line, you might prefer to have a short namespace that can be typed quickly.
- Its name must be unique in Cloud.

Next, start defining your container. The window opens to the **Single** container tab, by default.

Optional: Under the image name, select the tag or version of the image to use.

Select one of the Cloud spaces in your organization.

In the **Container name** field, enter the name for the container.

**Tip:**

The container name must start with a letter, and then can include uppercase letters, lowercase letters, numbers, periods (.), underscores (\_), or hyphens (-).

In the **Size** field, select a container size. The default size is **Micro (256 MB Memory, 16 GB Storage)**. The size that you choose impacts the amount of memory and disk space the container gets on the compute host during runtime, and cannot be changed after the container is created. Other available sizes are

- Pico (64 MB memory, 4 GB disk space)
- Nano (128 MB memory, 8 GB disk space)
- Tiny (512 MB memory, 32 GB disk space)
- Small (1 GB memory, 64 GB disk space)
- Medium (2 GB memory, 128 GB disk space)
- Large (4 GB memory, 256 GB disk space)
- X-Large (8 GB memory, 512 GB disk space)
- 2X-Large (16 GB memory, 1 TB disk space)

In the **Public IP address** field, choose whether to request and assign a new public IP address. Public IP addresses allow a public connection with the container. One instance when you might not want a public IP address is when you are using a database that is visible to the user's app only inside the private network.

**Note:**

Binding a public IP address does not include a SSL certificate which encrypts the data that are sent to and from your single container. If your app requires SSL encryption, you can either implement your own SSL solution or use a container group instead of a single container. Container groups are bound to a public route that already includes a SSL certificate, so you can access your container group with HTTPS without any additional configuration. For more information, see [Creating a container group with the GUI](#).

If you have an IP address you can use, select **Leave unassigned** and bind it to the container after the container is created.

In the **Public ports** field, specify any public ports to publish. The field is pre-populated with the ports that were specified in the Dockerfile for the image when the image was built. When you specify a public port, you can choose between UDP and TCP to indicate the IP protocol that you want to use. If you do not specify a protocol, your port is automatically exposed for TCP traffic. When you expose a public port, you create a Public Network Security Group for

your container that allows you to send and receive public data on the exposed port only. All other public ports are closed and cannot be used to access your app from the internet. You can include multiple ports by separating each port with a comma (,).

If a port is specified in the Dockerfile for the image that you are using, include that port.

#### Tip

- For the IBM certified Liberty Server image or a modified version of this image, enter the port 9080.
- For the IBM certified Node.js image or a modified version of this image, enter any port, such as port 8000.

Optional: If you want to add a volume, add environment variables, or bind a Cloud service, expand **Advanced Options**.

Optional: Define data or file storage. Associate an existing volume with a container or container group. A volume is a shared disk that allows files to be accessible from within a container. The path in the container indicates where in the container's file system the volume is mounted.

#### Restriction:

You can delete volumes from your Cloud space by using the command line only.

Click **Assign an existing volume**.

Select the volume from the drop-down list.

In the next field, specify a path on the container to map the volume. For example, `/var/lib/my_volume`

To specify that a volume is read-only, select **Read-only**. By default, volumes are read/write.

To add multiple volumes, click **Assign an existing volume** and repeat these steps.

Optional: Define environment variables.

Click **Add a new environment variable**.

Enter a name for the environment variable.

In the next field, specify a value for the environment variable.

To add multiple environment variables, click **Add a new environment variable** and repeat these steps.

Table 1. Suggested environment variables

| Environment variable                                                                                                                            | Description                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCS_BIND_APP=<br><appname>                                                                                                                      | Some Cloud services do not support direct binding to a container. In this case, you need to create a Cloud Foundry app and bind the Cloud service to it. Then, you bind the app to your container by using CCS_BIND_APP. The Cloud Foundry app acts as a bridge and allows Cloud to inject your bridge app's VCAP_SERVICES information into the running container instance. |
| CCS_BIND_SRV=<br><service_instance_name1>,<br><service_instance_name2>                                                                          | To bind a Cloud service directly to a container without using a bridge app, use CCS_BIND_SRV. This binding allows Cloud to inject the VCAP_SERVICES information into the running container instance. To list multiple Cloud services, include them as part of the same environment variable.                                                                                |
| <div><b>Note</b></div> <div>When a service does not support the use of the CCS_BIND_SRV= environment variable, use CCS_BIND_APP= instead.</div> |                                                                                                                                                                                                                                                                                                                                                                             |
| (Deprecated) CCS_SSH_KEY=<br><public_ssh_key>                                                                                                   | <div><b>This environment variable has been deprecated</b></div> <div>Use <code>bx ic exec</code> or <code>bx ic attach</code></div>                                                                                                                                                                                                                                         |

| Environment variable                          | Description                                                                                                                                                                                                                        |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                               | <div>for external access to your containers instead. For more information, see <a href="#">Logging in to a container with exec</a>.</div> <div>To add an SSH key to a container when you create it, you can use CCS_API_KEY.</div> |
| LOG_LOCATIONS=<br><i>&lt;path_to_file&gt;</i> | To add a log file to be monitored in the container, include the LOG_LOCATIONS environment variable with a path to the log file.                                                                                                    |

Optional: Define other services for the app. Bind a Cloud service from your Cloud space to your container. In the **Service binding** section, select a Cloud service instance and click **Add** to bind the service to your container. If you do not have any Cloud services added to the space yet, no services are displayed in the menu.

Deploy the container and wait for it to complete. Click **CREATE**. As the container is created, which might take a few moments, the container is also started in Cloud. Look in the **CONTAINER HEALTH** section for the container's status. When the creation is complete, the status is **Running**.

If you selected **Leave unassigned** for the **Public IP Address** because you have an existing IP address to use, bind the IP address to the container. For more information, see [Requesting and binding public IP addresses to containers](#).

If you are connecting to an external app from the container, there is up to a 5-minute wait after you started the container process before it can connect to the specified IP address. Then, you can verify that the app is functioning in the container by doing one of the following options:

You can open the app in a browser by creating a URL with the public IP address and the port.

Example

`http://public_IP:public_port`

### Note

If a service in the app exists that does not expose an HTTP endpoint, this option might not display the app in the browser.

From the Linux or UNIX command line, you can run `nc -zv <public_IP> <public_port>` to verify that a server is listening on the port that was specified.

Next, if you are an organization manager you can view your organization's container billing.

From your account details, in manage organizations, select an organization.

Select **Usage Dashboard** and review the usage and billing details.

## Running short-term tasks as single containers with the command line interface (CLI)

Create a container by using the `run` command. A container must include an image in it. If you do not have one yet, you can use one of the default IBM certified images that are available by default in your organization's private images registry.

Before you begin, consider the following steps.

You must have a Developer role for the Cloud space to create a container in it. For more information on the permissions of each roles and how to edit roles, see [Users and roles](#).

Identify an existing image from your organization's private images registry to use in your container by running the `bx ic images` command. To add an image to your registry, see [Adding Docker images to your organization's private Cloud images registry \(Deprecated\)](#).

Check to see whether an IP address for the space exists that you can use instead of creating a new one. For more information, see [Finding existing public IP addresses for your organization with the command line interface \(CLI\)](#).

If you know that you want to bind a Cloud service to the container, add the Cloud service to your space. For more information, see [Binding a service from the Cloud GUI](#).

If you know that you want a shared disk that allows files to be accessible from multiple

containers, create a volume before you create the container. For more information, see [Creating volumes with the command line \(CLI\)](#).

Create a container.

Create a container with an existing IP address and an existing image in your organization's private images registry.\*

```
bx ic run -p <ip-address>:<port_on_IP>:<public_port> -e <env_variable> -m
```

**Note:**

\*In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk (\*) in this topic.

Example

The following example creates a single container with these details.

- Name: *my\_container*
- Size: *64 MB*
- Image: *ibm/liberty*
- Mounted volume: *myvol*
- Exposed public port: *9080*
- Bound IP address: *192.0.2.23*
- Custom log file location: Standard Ubuntu log file. Logs from custom log file locations can be viewed in Kibana after the container is created.

```
bx ic run -p 192.0.2.23:9080:9080 -e "LOG_LOCATION=/var/log/dpkg.log" -m
```

As the container is created, it is also started. The container is deployed when the ID for the container is displayed.

Optional: List your running containers.\*

```
bx ic ps
```

If you are connecting to an external app from the container, there is up to a 5-minute wait after

you started the container process before it can connect to the specified IP address. Then, you can verify that the app is functioning in the container by doing one of the following options:

You can open the app in a browser by creating a URL with the public IP address and the port.

Example

```
http://public_IP:public_port
```

Note

If a service in the app exists that does not expose an HTTP endpoint, this option might not display the app in the browser.

From the Linux or UNIX command line, you can run `nc -zv <public_IP> <public_port>` to verify that a server is listening on the port that was specified.

Next, if you are an organization manager you can view your organization's container billing.

- Click on the user avatar.
- Click on **Account**.
- Expand your organization by clicking the + button.
- Select **Usage Dashboard** and review the usage and billing details.

For more information, go to [Managing your account](#).

## Updating short-term tasks in single containers with the command line interface (CLI)

To update an app, you must deploy a new container. If you push a new image, the container is not restarted automatically.

Table 2. Optional tools for updating an app

| Update tools | Description |
|--------------|-------------|
|--------------|-------------|



| Update tools                                                         | Description                                                                                                                                                                                                                                           |
|----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Delivery Pipeline                                                    | Automate your app builds and container deployments to Cloud by using the Delivery Pipeline, which is a Cloud service that is available to you. For more information, see <a href="#">Creating a single container by using the Delivery Pipeline</a> . |
| UrbanCode Deploy                                                     | Automate your app builds and container deployments to Cloud by using UrbanCode Deploy. For more information about purchasing UrbanCode Deploy, see the <a href="#">IBM UrbanCode Deploy</a> product page.                                             |
| Update the app yourself by using the IBM Cloud Container Service CLI | Complete the steps in this task to update the app yourself.                                                                                                                                                                                           |

To update a container yourself by using the IBM Cloud Container Service CLI:

Update the app locally.

Choose to either build your image directly in Cloud or build and test your image locally before you push it to Cloud.

- o To build the image directly in Cloud, run the following command.\*

#### Tip

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

```
bx ic build -t registry.DomainName/<my_namespace>/<image_name>:<tag> <
```

Example

```
bx ic build -t registry.DomainName/my_namespace/my_image:v1 .
```

#### Note

\*In this command, you can replace `bx ic` with `docker` when you [logged in to](#)

[IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk (\*) in this topic.

- To build the image locally and then push the image to Cloud, follow these steps.

If you are using the plug-in for IBM Cloud Container Service, log in again. Do not set the environment variables for option 2, so that docker commands will be sent to the Docker engine on your local machine.

```
bx ic init
```

Build the image locally from your Dockerfile.

```
docker build -t <image_name>:<tag> <dockerfile_location>
```

Example

```
docker build -t my_ibmliberty_image:v1 .
```

Run a container from the image to test that your new app runs locally by using the following command, where *Port* is the port for HTTP traffic.

```
docker run -d --name <container_name> <image_name>
```

Example

```
docker run -d --name my_container my_ibmliberty_image
```

If the app is running correctly, the container ID is displayed in the CLI output. To review the logs for the container, run `docker logs <container_name_or_id>`.

Tag the local image with your private Cloud registry and a new name. Use lowercase alphanumeric characters or underscores (\_) only in the image name. Other symbols, such as hyphens (-) or slashes (/), might prevent the image from being pushed to the image registry.

```
docker tag <current_image_name_or_ID>:<optional_tag> registry.
```

Example

```
docker tag my_ibmliberty_image registry.DomainName/<my_namespac
```

Push the image to your private Cloud registry by using the following command.

```
docker push registry.DomainName/<my_namespace>/<image_name>
```

Example

```
docker push registry.DomainName/my_namespace/my_ibmliberty_imag
```

### Important

When you push an image to your private Cloud registry, the size reported for the image is smaller than the size of the same image in your local Docker engine. The difference between the sizes does not indicate that an issue occurred when the image was pushed. The compressed size of the image is reported in IBM Cloud Container Service.

Unbind the public IP address from the container.

```
bx ic ip-unbind <public_IP_address> <container_name_or_id>
```

The container is no longer bound to the IP address, but the IP is still counted toward your organization's public IP quota.

Verify that the IP address was unbound by running the following command.\*

```
bx ic inspect <container_name_or_id>
```

If the value for `PublicIpAddress` is not empty, wait a few minutes longer and run the command again before you continue with the next step.

After the IP address was removed from the existing container, create another container with the updated image.\*

```
bx ic run [-p PORT] [-P] [-d] [-e ENV] [--env-file
ENVFILE] [-it] [--link NAME:ALIAS] [-m MEMORY] --name NAME [--volume VOLU
```

Bind the IP address to the new container.

```
bx ic ip-bind <public_IP_address> <container_name_or_id>
```

Optional: Delete the old container.\*

```
bx ic rm <container_name_or_id>
```

## Linking single containers with the command line interface (CLI)

When a container is linked to another container in the same space, IBM Cloud Container Service creates a host entry on the recipient container for the source container. You can use an alias to refer to the source container, rather than a specific IP address that might change frequently.

Whenever you want a single container to communicate with another container that is running an image, such as `dbimage`, you can address the container by using an alias for the host name, such as `dbserver`. With this capability, you can create a `webserver` image that refers to the database server image by using the `dbserver` host name, regardless of what its IP address or actual name is. You can run many instances of the pair in the same image without having to hand-off configuration or IP information. You can always refer to the database server as `dbserver`.

Create a source container by running the following command.\*

**Tip:**

Make sure to include the `-p` option to publish the port, which exposes the port number to the recipient container. In a Dockerfile, the port that is exposed in the image with the `EXPOSE` statement is not currently used by the linking run time.

**Tip:**

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

```
bx ic run -p <port> --name <source_container_name> registry.DomainName/<m
```

## Example

```
bx ic run -p 17546 --name source registry.DomainName/<my_namespace>/dbima
```

### Note:

\*In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk (\*) in this topic.

Create a recipient container and link it to the source container by running the following command. \* For the alias, select a custom name for the image that is used in the source container. You can expose multiple ports for the recipient container to the host container by including the `-p` option multiple times.

```
bx ic run -p <port> -p <port> --name <recipient_container_name> --link <s
```

## Example

```
bx ic run -p 12534 -p 15256 --name recipient --link source:dbserver regis
```

The container is linked when the ID for the recipient container is displayed.

Optional: Verify the environment variables that are injected by the link by logging into your container using `exec`.\*

```
bx ic exec -it <recipient_container_name> bash
```

Run the `env` command in the bash window to see the details of the alias.

```
"Env": [
 "<alias>_PORT_<source_container_port>/tcp_TCP_PORT=<source_container_
 "<alias>_PORT_<source_container_port>/tcp_TCP=tcp://<recipient_contai
 "<alias>_ENV_space_id=<space_ID>",
 "logging_password=",
 "space_id=<space_ID>",
 "logstash_target=logs.opvis.bluemix.net:9091",
 "<alias>_PORT_<source_container_port>/tcp_TCP_ADDR=<recipient_contain
 "<alias>_ENV_metrics_target=metrics.opvis.bluemix.net:9095",
 "<alias>_NAME=/<recipient_container>/<alias>",
 "<alias>_ENV_logstash_target=logs.opvis.bluemix.net:9091",
 "<alias>_ENV_logging_password=",
 "<alias>_PORT_<source_container_port>/tcp_TCP_PROTO=tcp",
```

```
"metrics_target=metrics.opvis.bluemix.net:9095"
],
```

## Requesting and binding public IP addresses to containers

Request and bind a public IP address to a container in Cloud.

### Note

Binding a public IP address does not include a SSL certificate which encrypts the data that are sent to and from your single container. If your app requires SSL encryption, you can either implement your own SSL solution or use a container group instead of a single container. Container groups are bound to a public route that already includes a SSL certificate, so you can access your container group with HTTPS without any additional configuration. For more information, see [Creating a container group with the GUI](#).

Before you begin, to check whether a public IP address exists for the space that you can use instead of creating one, see [Finding existing public IP addresses for your organization with the command line interface \(CLI\)](#).

Request a public IP address. When requesting a public IP address, a random one from the public IP address pool is allocated to the space. While allocated to a space, the IP address does not change.

```
bx ic ip-request
```

Note the IP address.

Bind the address to a new container or an existing running container.

- o To bind to an existing running container

```
bx ic ip-bind <public_IP_address> <container_name_or_id>
```

Example

```
bx ic ip-bind 192.0.2.23 my_container
```

- To bind to a new container

```
bx ic run -p <ip-address>:<container-port>:<container-port> --name <container-name>
```

The bind request is made immediately, but the connectivity to the container with the IP address might take a few minutes.

## Finding existing public IP addresses for your organization with the command line interface (CLI)

To make single containers accessible from the Internet, you must bind a public IP address to it. Before you request an IP address, review the list of public IP addresses already assigned to your organization and see whether one of those existing IP addresses are available.

### Note

Binding a public IP address does not include a SSL certificate which encrypts the data that are sent to and from your single container. If your app requires SSL encryption, you can either implement your own SSL solution or use a container group instead of a single container. Container groups are bound to a public route that already includes a SSL certificate, so you can access your container group with HTTPS without any additional configuration. For more information, see [Creating a container group with the GUI](#).

List available public IP addresses for your space.

```
bx ic ips
```

If you do not have any IP addresses or want to request one, you can choose between the following options. You can request one either when you create a single container by using the Cloud GUI or when you run `bx ic ip-request` from the command line. To request an IP address by using the command line, see [Requesting and binding public IP addresses to containers](#).

## Unbinding public IP addresses from a container

You can unbind a public IP address from a container, but the IP address is still available to other

containers and counted toward your organization's public IP quota.

Unbind the public IP address from the container.

```
bx ic ip-unbind <public_IP_address> <container_name_or_id>
```

The container is no longer bound to the IP address, but the IP is still counted toward your organization's public IP quota.

Verify that the IP address is successfully unbound.

```
bx ic ips
```

## Removing public IP addresses from your quota

If you are not using a public IP address or must delete one to request a new one, you can remove a public IP address from your organization's quota.

Before you begin, unbind the public IP address from any containers that are using it.

Remove the IP address from your organization's quota.

```
bx ic ip-release <public_IP_address>
```

Optional: Verify that the IP address is not listed as allocated anymore.

```
bx ic ips
```

## Removing single containers

To maximize the use of your quota, remove containers that are not in use occasionally.

Remove a container by using one of the following methods.

- From the Cloud GUI

From the Cloud Dashboard, select the container that you want to delete.

Expand the **More actions...** menu and click **Delete**.

- In the tile for the container, click the gear icon and click **Delete container**.

- From the CLI \*

```
bx ic rm [-f] CONTAINER [CONTAINER]
```



### Note

\*In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk (\*) in this topic.

Optional: Verify that the container was removed by running the following command and confirming that the container does not appear in the list.\*

```
bx ic ps -a
```

# Running highly available processes as container groups (Deprecated)

Highly available containers means less downtime for your app. The more widely you distribute your container setup, the less likely your users are to experience downtime with your app. You can achieve higher availability by using IBM® Cloud Container Service built-in capabilities to increase resiliency against potential failure conditions, such as host failures, network failures, or application software failures.

## Attention

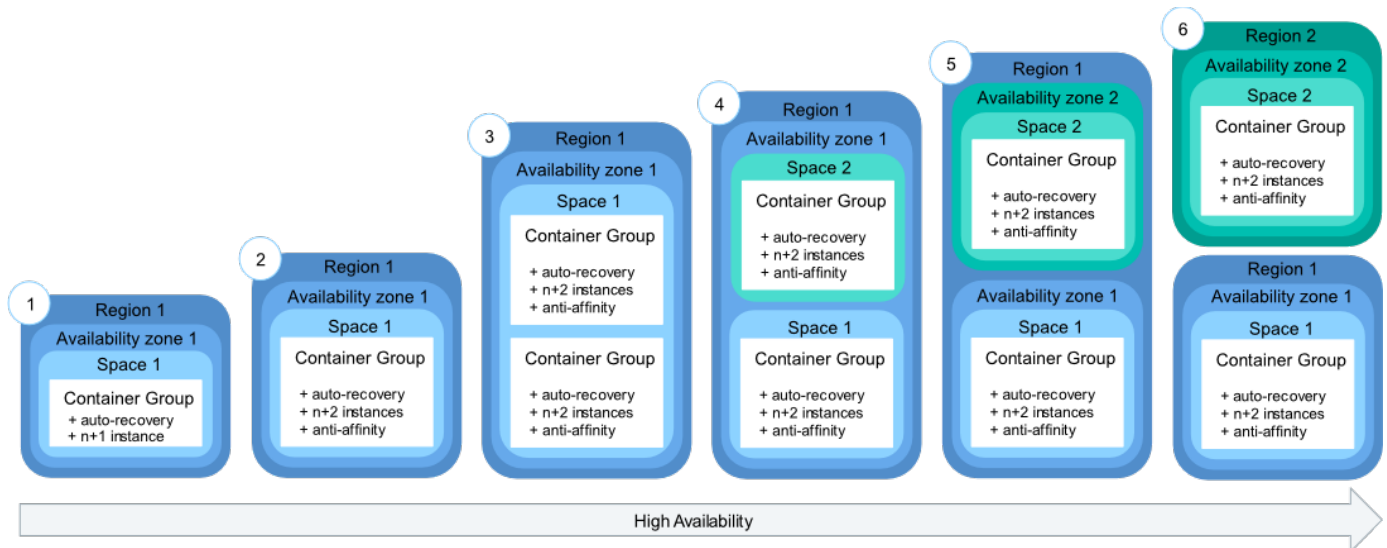
Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

In this page:

- [Options for setting up container groups](#) by availability
- Running long-term services as container groups from the [GUI](#) or the [CLI](#)
- Automatically [scaling groups](#) (deprecated)
- [Moving groups](#) to another space
- Updating an app that runs in your container group by using the [GUI](#) or the [CLI](#)
- Running highly available groups [in the same space](#) or [different regions](#)
- Connecting to a [private group](#)
- [Removing](#) groups

## Options for setting up container groups by availability

Review these potential container group set ups that are ordered with increasing degrees of availability:



One container group with auto-recovery and n+1 instance pattern

One container group with auto-recovery, n+2 instances pattern, and anti-affinity

Two container groups with auto-recovery, n+2 instances pattern, and anti-affinity that run in the same space and share the same route

Two container groups with auto-recovery, n+2 instances pattern, and anti-affinity that run in different spaces and share the same route

Two container groups with auto-recovery, n+2 instances pattern, and anti-affinity that run in different IBM Cloud Container Service availability zones and share the same route

Two container groups with auto-recovery, n+2 instances pattern, and anti-affinity that run in different regions and share the same route

Learn more about how you can use these techniques to increase the availability of your app:

## Deploy apps in container groups rather than single containers

A container group includes two or more containers that run the same image. Use container groups for running long-term services with workloads that require scalability and reliability or for testing at the required scale. You can create container groups from either the [Cloud GUI](#) or the [CLI](#).

```
bx ic group-create -p 9080 -n mylibertyhost -d AppDomainName --name
```

## Enable automatic recovery in the container group

When automatic recovery is enabled for a container group and the group has been running for 10 minutes, the group's load balancer starts to regularly check the health of each container instance in the group via HTTP requests. If a container instance

does not respond within 100 seconds, it is marked as inactive. Inactive container instances are removed from the group and re-created by auto-recovery.

Auto-recovery attempts to recover container instances in a group 3 times. After the third attempt, auto-recovery does not recover any container instances in the group for 60 minutes. After 60 minutes, the auto-recovery process starts again. You can create container groups with automatic recovery from the [Cloud GUI](#) or the [CLI](#).

```
bx ic group-create -p 9080 --auto -n mylibertyhost -d AppDomainName
```

### **Include enough instances for your app's workload, plus two**

To make your container group highly available and more resilient to failure, consider including extra instances than the minimum to handle the expected workload. Extra instances can handle the workload in case another instance crashes. To ensure automatic recovery of a failed instance without affecting the workload, include one extra instance. For protection against two simultaneous failures, include two extra instances. This set up is an  $N+2$  pattern, where  $N$  is the number of instances to handle the requests and  $+2$  is an extra two instances. You can specify how many instances to include when you create a container group from the [Cloud GUI](#) or the [CLI](#).

```
bx ic group-create -p 9080 --desired 5 --auto -n mylibertyhost -d Ap
```

### **Spread container instances across compute nodes**

When you create a container group, each instance in the group might be deployed to the same physical compute node. This setup where container instances exist on the same compute node is known as affinity or co-location. When you create the container group from the [CLI](#), you can spread the container instances across different physical compute nodes to increase your app's availability by using the anti-affinity option (`--anti`).

#### **Note**

To use the `--anti` option from the CLI, you must have installed the IBM Cloud Container Service plug-in (`bx ic`) version 0.8.934 or later.

```
bx ic group-create -p 9080 --anti --desired 5 --auto -n mylibertyhos
```

### **Create multiple container groups that use the same route**

You can create two container groups from the same image in the same Cloud space that use the same route. If an error occurs where the instances in one container group are not reachable, user traffic is routed automatically to the second container group. For more information about sharing a route between container groups, see [Running highly available container groups in the same space](#).

```
bx ic group-create -p 9080 --anti --desired 5 --auto -n mylibertyhos
```

```
bx ic group-create -p 9080 --anti --desired 5 --auto -n mylibertyhos
```

### Spread container groups across IBM Cloud Container Service availability zones

An IBM Cloud Container Service availability zone is a location within a region that IBM Cloud Container Service runs in. When IBM Cloud Container Service is used for the first time in a Cloud space, containers are created in the default IBM Cloud Container Service availability zone for that region.

*Table 1. IBM Cloud Container Service availability zones*

| Region         | Data center location | IBM Cloud Container Service availability zone names |
|----------------|----------------------|-----------------------------------------------------|
| US South       | Dallas 1             | dal09-01                                            |
| US South       | Dallas 2             | dal09-02                                            |
| US South       | Dallas 3             | dal10-03                                            |
| United Kingdom | London               | lon02-01                                            |
| United Kingdom | Amsterdam            | ams03-01                                            |

#### Important

After configuring your spaces with different IBM Cloud Container Service availability zones, you might use these commands to create the container groups.

IBM Cloud Container Service availability zone 1:

```
bx ic group-create -p 9080 --anti --desired 5 --auto -n mylibertyhos
```

IBM Cloud Container Service availability zone 2:

```
bx ic group-create -p 9080 --anti --desired 5 --auto -n mylibertyhos
```

## Spread container groups across regions

When you spread container groups across Cloud regions, you can allow load balancing to occur based on the region the user is in. If the container group, or hardware, or even an entire datacenter in one region goes down, traffic is routed to the container group that is deployed in another region.

To deploy container groups in multiple regions, see [Running highly available container groups in different Cloud regions](#).

### Important

After configuring your custom domain, you might use these commands to create the container groups.

Region 1:

```
bx ic group-create -p 9080 --anti --desired 5 --auto -n mylibertyhos
```

Region 2:

```
bx ic group-create -p 9080 --anti --desired 5 --auto -n mylibertyhos
```

## Running long-term services as container groups from the Cloud GUI

Create and deploy a scalable group container from the Cloud GUI. A container group includes two or more containers that run the same image. Use container groups for running long-term services with workloads that require scalability and reliability or for testing at the required scale.

Before you begin, consider the following prerequisites.

You must have a Developer role for the Cloud space to create a container in it. For more information on the permissions of each roles and how to edit roles, see [Users and roles](#).

From the Cloud GUI, you create a container from either the IBM-provided public images, or from an existing image in your organizations private Cloud registry. For more information about adding an image to your Catalog, see [Adding Docker images to your organization's private Cloud images registry \(Deprecated\)](#).

If you know that you want to bind a Cloud service to the container, add the Cloud service to your space. For more information, see [Binding a service from the Cloud GUI](#).

(Optional) If you know that you want a shared disk that allows files to be accessible from within a container group, create a volume before you create the container group. For more information, see [Creating volumes with the command line \(CLI\)](#).

To create and deploy a container group:

From the catalog, select **Containers** and choose an image. Both images that are provided by IBM and images that are stored in your private Cloud registry are displayed.

If a namespace is already set for your organization, you are not prompted to create one and can continue with the next step. If a namespace is not specified for your organization yet, you are prompted to set one for the image registry. A namespace is a unique name to identify your private registry in Cloud. The namespace is assigned one time for an organization and cannot be changed after it is created. If you want to use IBM Cloud Container Service in multiple Cloud regions, you must set up a namespace for each region.

**Tip:**

Observe the following rules for the namespace:

- It can contain only lowercase letters, numbers, or underscores (\_).
- It can be 4 - 30 characters. If you plan to manage containers from the command line, you might prefer to have a short namespace that can be typed quickly.
- Its name must be unique in Cloud.

Select one of the Cloud spaces in your organization.

Next, start defining your container group. The window opens to the **Single** tab, by default, so select the **Scalable** tab to create a container group.

Optional: Under the image name, select the tag or version of the image to use.

In the **Container group name** field, enter a name for the container.

**Tip:**

The container name must start with a letter, and then can include uppercase letters, lowercase letters, numbers, periods (.), underscores (\_), or hyphens (-).

In the **Instances** field, specify the number of instances in the group. The default is **2**. Each instance is the same size and must use the same image. The number of container group instances that your app requires depends on the expected workload and the time in which a request can be processed by the app. For example, multi-tasking or network connectivity affects how many instances might be required for your app.

To determine the number of desired instances, deploy a single container in Cloud that runs your app and perform a load test on this container. If the app can handle, for example, 100 requests per second, but you are expecting an average workload of 300 requests per second, then the desired number of instances for the container group is 3.

To make your container group highly available and more resilient to failure, consider including extra instances than the minimum to handle the expected workload. Extra instances can handle the workload in case another instance crashes. To ensure automatic recovery of a failed instance without affecting the workload, include one extra instance. For protection against two simultaneous failures, include two extra instances. This set up is an  $N+2$  pattern, where  $N$  is the number of instances to handle the requests and  $+2$  is an extra two instances. If you find you need more instances after the group is deployed, you can add more instances without having to re-create the group.

**Important:**

A container group should include at least 2 container instances. If you include only 1 instance in the group and that instance becomes unavailable, your users can experience downtime for the app.

In the **Size** field, select a container size. The default size is **Micro (256 MB Memory, 16 GB Storage)**. The size that you choose impacts the amount of memory and disk space the container gets on the compute host during runtime, and cannot be changed after the container is created. Other available sizes are

- Pico (64 MB memory, 4 GB disk space)
- Nano (128 MB memory, 8 GB disk space)
- Tiny (512 MB memory, 32 GB disk space)
- Small (1 GB memory, 64 GB disk space)



- Medium (2 GB memory, 128 GB disk space)
- Large (4 GB memory, 256 GB disk space)
- X-Large (8 GB memory, 512 GB disk space)
- 2X-Large (16 GB memory, 1 TB disk space)

Select the full public URL. This URL is also called a route. By using a host and domain for the container group rather than an IP address for a single container instance, you are providing load balancing for your app. Your container group might not be immediately available at the public URL upon building as network connectivity can take up to 5 minutes to resolve.

In the **Host** field, specify the host name, such as *mycontainerhost*. Do not include underscores ( `_` ) in the host name.

For the system **Domain**, select AppDomainName. The system domain already provides an SSL certificate, so you can access your container group with HTTPS without any additional configuration. You can also select a custom domain that you created for your organization. To create a custom domain, you must register the custom domain on a public DNS server, configure the custom domain in Cloud, and then map the custom domain to the Cloud system domain on the public DNS server. After your custom domain is mapped to the Cloud system domain, requests for your custom domain are routed to your application in Cloud. When you create a custom domain, do not include underscores ( `_` ) in the domain name. For more information about custom domains, see [Creating and using a custom domain](#). To make your custom domain secure, [upload a SSL certificate](#), so your container groups can be accessed with HTTPS.

The route domain and the host combined form the full public route URL, such as `http://mycontainerhost.AppDomainName` and must be unique within Cloud.

In the **HTTP port** field, specify a public port to publish. You can type a port or select a port from drop-down list. When you type a port, you can choose between UDP and TCP to indicate the IP protocol that you want to use. If you do not specify a protocol, your port is automatically exposed for TCP traffic. The drop-down list was pre-populated with the ports that were specified in the Dockerfile for the image when the image was built. For container groups, you cannot include multiple ports. When you bind a route, containers in your group must listen for HTTP traffic on the group's exposed port. Non-HTTP ports cannot be exposed publicly. When HTTPS traffic arrives on the exposed port, the (Go)Router completes the

HTTPS termination. Then, the HTTP protocol is used on the private virtual network between the (Go)Router and the containers.

If a port is specified in the Dockerfile for the image that you are using, include that port.

#### Tip

- For the IBM certified Liberty Server image or a modified version of this image, enter the port 9080.
- For the IBM certified Node.js image or a modified version of this image, enter any port, such as port 8000.

Optional: Enable auto-recovery. Select **Enable automatic recovery**. Automatic recovery removes unavailable container instances and replaces them with new container instances. When automatic recovery is enabled for a container group and the group has been running for 10 minutes, the group's load balancer starts to regularly check the health of each container instance in the group via HTTP requests. If a container instance does not respond within 100 seconds, it is marked as inactive. Inactive container instances are removed from the group and re-created by auto-recovery. Auto-recovery attempts to recover container instances in a group 3 times. After the third attempt, auto-recovery does not recover any container instances in the group for 60 minutes. After 60 minutes, the auto-recovery process starts again.

Optional: Define data or file storage. Associate an existing volume with a container or container group. A volume is a shared disk that allows files to be accessible from within a container. The path in the container indicates where in the container's file system the volume is mounted.

#### Restriction:

You can delete volumes from your Cloud space by using the command line only.

Click **Assign an existing volume**.

Select the volume from the drop-down list.

In the next field, specify a path on the container to map the volume. For example, `/var/lib/my_volume`

To specify that a volume is read-only, select **Read-only**. By default, volumes are read/write.

To add multiple volumes, click **Assign an existing volume** and repeat these steps.

Optional: Define environment variables.

Click **Add a new environment variable**.

Enter a name for the environment variable.

In the next field, specify a value for the environment variable.

To add multiple environment variables, click **Add a new environment variable** and repeat these steps.

*Table 2. Suggested environment variables*

| Environment variable                                                   | Description                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCS_BIND_APP=<br><appname>                                             | Some Cloud services do not support direct binding to a container. In this case, you need to create a Cloud Foundry app and bind the Cloud service to it. Then, you bind the app to your container by using CCS_BIND_APP. The Cloud Foundry app acts as a bridge and allows Cloud to inject your bridge app's VCAP_SERVICES information into the running container instance. |
| CCS_BIND_SRV=<br><service_instance_name1>,<br><service_instance_name2> | To bind a Cloud service directly to a container without using a bridge app, use CCS_BIND_SRV. This binding allows Cloud to inject the VCAP_SERVICES information into the running container instance. To list multiple Cloud services, include them as part of the same environment variable.                                                                                |

**Note**

When a service does not support the use of the CCS\_BIND\_SRV= environment variable, use

| Environment variable                                       | Description                                                                                                                                                                                                                                                                                                                                                  |
|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CCS_BIND_APP= instead.                                     |                                                                                                                                                                                                                                                                                                                                                              |
| (Deprecated) CCS_SSH_KEY=<br><i>&lt;public_ssh_key&gt;</i> | <div> <p><b>This environment variable has been deprecated</b></p> <p>Use <code>bx ic exec</code> or <code>bx ic attach</code> for external access to your containers instead. For more information, see <a href="#">Logging in to a container with exec</a>.</p> </div> <p>To add an SSH key to a container when you create it, you can use CCS_API_KEY.</p> |
| LOG_LOCATIONS=<br><i>&lt;path_to_file&gt;</i>              | To add a log file to be monitored in the container, include the LOG_LOCATIONS environment variable with a path to the log file.                                                                                                                                                                                                                              |

Optional: Define other services for the app. Bind a Cloud service from your Cloud space to your container. In the **Service binding** section, select a Cloud service instance and click **Add** to bind the service to your container. If you do not have any Cloud services added to the space yet, no services are displayed in the menu.

Deploy the container and wait for it to complete. Click **CREATE**. As the container is created, which might take a few moments, the container is also started in Cloud. Look in the **CONTAINER HEALTH** section for the container's status. When the creation is complete, the status is **Running**.

If you are connecting to an external application from the container, there is up to a 5-minute wait after running the container process before it can connect to the specified route. Then, you can verify that the app is running in the container group by opening

`https://<host>.AppDomainName` in a browser.

The more widely you distribute your container setup, the less likely your users are to experience downtime with your app. After you create a container group, create another with the same image in the same space, another space in the same organization, or in another region to make your app more widely distributed. For more information, see the following topics.

[Running highly available processes as container groups \(Deprecated\)](#)

[Running highly available container groups in different Cloud regions](#)

To increase or decrease the number of container instances that run in your container group, open the container group in the Cloud GUI. To increase the number of instances, click the **+** button, to decrease the number, click the **-** button. Then, click **Save**.

## Running long-term services as container groups with the command line interface (CLI)

Create and deploy a scalable group container from the IBM Cloud Container Service CLI. A container group includes two or more containers that run the same image. Use container groups for running long-term services with workloads that require scalability and reliability or for testing at the required scale.

Before you begin, consider the following steps.

You must have a Developer role for the Cloud space to create a container in it. For more information on the permissions of each roles and how to edit roles, see [Users and roles](#).

Identify an existing image from your organization's private images registry to use in your container by running the `bx ic images` command. To add an image to your registry, see [Adding Docker images to your organization's private Cloud images registry \(Deprecated\)](#).

If you know that you want to bind a Cloud service to the container, add the Cloud service to your space. For more information, see [Binding a service from the Cloud GUI](#).

If you know that you want a shared disk that allows files to be accessible from multiple containers, create a volume before you create the container. For more information, see [Creating volumes with the command line \(CLI\)](#).

Create a scalable container group.

Create a container group with an existing image in your organization's private images registry by running one of the following `group-create` commands.

```
bx ic group-create --anti --auto --desired <instances> -e "LOG_LOCATIONS=
```

#### Example

The following example creates a container group with 3 instances that is named *my\_group* from the **ibm/liberty** image. The anti-affinity and auto-recovery feature is enabled for the group to assure higher availability for the app. To access the group from the internet, the public port 9080 is exposed and the route `myhost.AppDomainName` is mapped to the group. The volume *myvol* is mounted to the group to persist container data and the standard Ubuntu log file is added as a custom log file location. Logs from custom log file locations can be viewed in Kibana after the group is created.

```
bx ic group-create --anti --auto --desired 3 -e "LOG_LOCATIONS=/var/log/d
```

As the container group is created, it is also started. The container group is deployed when the ID for the group is displayed.

Optional: List your running container groups.

```
bx ic groups [-q]
```

If you are connecting to an external application from the container, there is up to a 5-minute wait after running the container process before it can connect to the specified route. Then, you can verify that the app is running in the container group by opening `https://<host>.AppDomainName` in a browser.

The more widely you distribute your container setup, the less likely your users are to experience downtime with your app. After you create a container group, create another with the same image in the same space, another space in the same organization, or in another region to make your app more widely distributed. For more information, see the following topics.

[Running highly available processes as container groups \(Deprecated\)](#)

[Running highly available container groups in different Cloud regions](#)

After the container group is created, you can increase and decrease the number of instances that run in the container group as well as add further environment variables. For further

information, review the `bx ic group-update` command.

## Automatically scaling container groups (deprecated)

The Auto-Scaling for Containers capability is being deprecated. You can replace the functions by using IBM Cloud Container Service. For details, see [Auto-Scaling Retirement](#).

Are you looking for information about scaling Cloud Foundry applications? Check out [IBM Auto-Scaling for IBM Cloud](#).

## Migrating to Kubernetes Autoscaling


With [Kubernetes](#), you can enable [Horizontal Pod Autoscaling](#) to scale your apps based on CPU.

Install the required [CLIs](#).

Create a cluster by following the steps in [Getting started with Kubernetes clusters in IBM Cloud](#). Be sure to set up your cluster to replicate your current container group, including logging, monitoring, vulnerability advisor, and any other customizations you have in place. [Set the context](#) for your cluster.

Deploy your existing app to your cluster [from the CLI](#). When you deploy your app, you must request CPU.

```
kubectl run <name> --image=<image> --requests=cpu=<cpu> --expose --port=<
```

|                                                                                                                                    |                                                                                                         |
|------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
|  <b>Understanding this command's components</b> |                                                                                                         |
| --image                                                                                                                            | The application that you want to deploy.                                                                |
| --requests=cpu                                                                                                                     | The required CPU for your container, which is specified in milli-cores. As an example, --requests=200m. |
| --expose                                                                                                                           | When true, creates an external service.                                                                 |
| --port                                                                                                                             | The port where your app is available externally.                                                        |

**Note:**

For more complex deployments, you may need to create a [deployment script](#).

Create a Horizontal Pod Autoscaler and define your policy. For more information about working with the `kubectl autoscale` command, see [the Kubernetes documentation](#).

```
kubectl autoscale deployment <deployment_name> --cpu-percent=<percentage>
```



### Understanding this command's components

|                            |                                                                                                                      |
|----------------------------|----------------------------------------------------------------------------------------------------------------------|
| <code>--cpu-percent</code> | The average CPU utilization that is maintained by the Horizontal Pod Autoscaler, which is specified as a percentage. |
| <code>--min</code>         | The minimum number of deployed pods that are used to maintain the specified CPU utilization percentage.              |
| <code>--max</code>         | The maximum number of deployed pods that are used to maintain the specified CPU utilization percentage.              |

Verify that your application is working correctly.

- Check your service dashboard to ensure that your app is running.
- Check your app to be sure that key changes were implemented.

[Deprovision and remove](#) your original instance of Containers.

## Moving container groups to other Cloud spaces

If you must move a container group to another space in the same organization, you can create a second container group with the same route. Then, you can remove the first group without causing any downtime for your app.

When you map an existing route to a new container group, the references of the container instances are added to the load balancer for the space. As a consequence, the load balancer considers the new container group when it routes incoming traffic. If you delete a container group, the load balancer automatically removes the related container instance references.

Log in to the Cloud Foundry CLI. Enter your Cloud user name and password when prompted.

```
bx login [-a api.DomainName] [-sso]
```



The single-sign-on parameter `--sso` is required when you log in with a federated ID. If this option is used, open <https://login.ng.bluemix.net/UAALoginServerWAR/passcode> to obtain a one-time passcode. If you do not include the `--sso` option, complete these substeps.

For **Email**, enter your IBMId for Cloud.

For **Password**, enter the password for the IBMId. Your Cloud organizations and spaces are retrieved.

Enter the number that represents one of your Cloud organizations.

Enter the number that represents one of your existing Cloud spaces.

Initialize IBM Cloud Container Service.

```
bx ic init
```

Create a container group and map a route to it. The following command creates a container group that is named *my\_group\_1* from the **ibmliberty** image. The container group is accessible by using the public route `myhost.AppDomainName`.

**Note:**

The host name, in this example *myhost*, needs to be unique across all organizations in Cloud. As a consequence, you need to select a new name for *myhost* to move forward.

```
bx ic group-create -p 9080 -n myhost -d AppDomainName --name my_group_1 r
```

Verify that your container group was created.

```
bx ic groups
```

Change your organization space. Replace *SPACE* with the name of the space where you want to move your container group to.

```
cf target -s SPACE
```

Reinitialize your IBM Cloud Container Service to point to the new space.

```
bx ic init
```

Create another container group in the new space and map the existing route to it. The following command creates a container group that is named *my\_group\_2* from the **ibmliberty** image. It is mapped to the same public route as the first container group you created.

**Note:**

The name of a container group needs to be unique within an organization space. You can

reuse the name when you create a new container group in another space of your organization.

```
bx ic group-create -p 9080 -n myhost -d AppDomainName --name my_group_2 r
```

Verify that your container group was created.

```
bx ic groups
```

Now, that you created a new container group in your new space and mapped the existing route to it, you can go ahead and remove the first container group.

Change to the organization space where you created the first container group. Replace *SPACE* with the name of your space.

```
cf target -s SPACE
```

Reinitialize your IBM Cloud Container Service to point to the selected space.

```
bx ic init
```

Unmap the route from the container group. Replace *myhost* with the name of the host you selected. When you unmap the route from your container group, all container references are deleted from the load balancer and not considered when incoming traffic is routed.

```
bx ic route-unmap -n myhost -d AppDomainName my_group_1
```

Optional: Remove the container group.

```
bx ic group-remove my_group_1
```

Optional: Verify that your container group was removed.

```
bx ic groups
```

## Updating an app that runs in your container group by using the Cloud GUI

To update an app that is running in a container group, you must update your image and deploy a new container group from it. During the update process, you are not required to stop the old running container group. You can create a new container group, map the existing route information to it, and then unmap the route from the outdated container group. Your new app is deployed without any downtime.

Table 3. Optional tools for updating an app

| Update tools                                                                                            | Description                                                                                                                                                                                                                                         |
|---------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Delivery Pipeline                                                                                       | Automate your app builds and container deployments to Cloud by using the Delivery Pipeline, which is a Cloud service that is available to you. For more information, see <a href="#">Creating container groups by using the Delivery Pipeline</a> . |
| UrbanCode Deploy                                                                                        | Automate your app builds and container deployments to Cloud by using UrbanCode Deploy. For more information about purchasing UrbanCode Deploy, see the <a href="#">IBM UrbanCode Deploy</a> product page.                                           |
| Update the app yourself by using a combination of the IBM Cloud Container Service CLI and the Cloud GUI | Complete the steps in this task to update the app yourself.                                                                                                                                                                                         |

To update your app yourself by using the IBM Cloud Container Service CLI and the Cloud GUI:

Change to the directory where your Dockerfile and app code is saved.

Update your app locally.

Choose to either build your image directly in Cloud or build and test your image locally before you push it to Cloud.

- o To build the image directly in Cloud, run the following command.\*

#### Tip

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

```
bx ic build -t registry.DomainName/<my_namespace>/<image_name>:<tag> <
```

Example

```
bx ic build -t registry.DomainName/my_namespace/my_image:v1 .
```

## Note

\*In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk (\*) in this topic.

- o To build the image locally and then push the image to Cloud, follow these steps.

If you are using the plug-in for IBM Cloud Container Service, log in again. Do not set the environment variables for option 2, so that `docker` commands will be sent to the Docker engine on your local machine.

```
bx ic init
```

Build the image locally from your Dockerfile.

```
docker build -t <image_name>:<tag> <dockerfile_location>
```

Example

```
docker build -t my_ibmliberty_image:v1 .
```

Run a container from the image to test that your new app runs locally by using the following command, where *Port* is the port for HTTP traffic.

```
docker run -d --name <container_name> <image_name>
```

Example

```
docker run -d --name my_container my_ibmliberty_image
```

If the app is running correctly, the container ID is displayed in the CLI output. To review the logs for the container, run `docker logs <container_name_or_id>`.

Tag the local image with your private Cloud registry and a new name. Use

lowercase alphanumeric characters or underscores (\_) only in the image name. Other symbols, such as hyphens (-) or slashes (/), might prevent the image from being pushed to the image registry.

```
docker tag <current_image_name_or_ID>:<optional_tag> registry.
```

Example

```
docker tag my_ibmliberty_image registry.DomainName/<my_namespac
```

Push the image to your private Cloud registry by using the following command.

```
docker push registry.DomainName/<my_namespace>/<image_name>
```

Example

```
docker push registry.DomainName/my_namespace/my_ibmliberty_imag
```

### Important

When you push an image to your private Cloud registry, the size reported for the image is smaller than the size of the same image in your local Docker engine. The difference between the sizes does not indicate that an issue occurred when the image was pushed. The compressed size of the image is reported in IBM Cloud Container Service.

Verify that the image was pushed to your Cloud registry.\*

```
bx ic images
```

Optional: Review image vulnerabilities.

From the Cloud GUI, go to the catalog and select **Containers**.

Click the image that you added.

In the **Vulnerability Assessment** section, review the status of your vulnerability assessment. The status is displayed as one of the following conditions:

- Safe to Deploy - No significant vulnerabilities were found.
- Deploy with Caution - Significant vulnerabilities were found to be addressed.
- Deployment Blocked - Significant vulnerabilities were found and must be addressed before the image can be deployed.
- Incomplete - The scan is not complete. The scan might still be running or the image's operating system might not be compatible. Wait and try the scan again. If the scan still does not complete, push the image again to start a new scan. Images with incomplete scans are not blocked for deployment.

Note the route information that is mapped to the running container group.

From the Cloud GUI, select the container group.

Note the route. The host and the domain combine to form the full public route URL, such as `http://mycontainerhost.AppDomainName`.

Create another scalable container group and include the route.

From the catalog, select **Containers** and choose an image.

Next, start defining your container group. By default, the window opens to the **Single** tab. So select the **Scalable** tab to create a container group. As you define your container group, enter the hostname and domain that you retrieved from the previous step in the Host and Domain fields.

Review the [Running long-term services as container groups from the Cloud GUI](#) topic for more information about how to create a container group.

Click **CREATE** to deploy your container group from the updated image. During the deployment process, the container group is also started.

If you are connecting to an external application from the container, there is up to a 5-minute wait after running the container process before it can connect to the specified route. Then, you can verify that the app is running in the container group by opening `https://<host>.AppDomainName` in a browser.

Unmap route from the outdated container group.

From the Cloud GUI, select the outdated container group.

In the **Group details** section, edit the route.

Change the host name. If you have multiple domain names, you can also select a new domain name for your container group.

Click **SAVE** to save your changes.

**Note:**

The route unmapping might take up to five minutes to occur.

Optional: Delete the outdated container group.

From the Cloud GUI, select the outdated container group.

From the **More actions...** menu, click **Delete** to remove the container group.

## Updating an app that runs in your container group by using the command line

To update an app that is running in a container group, you must update your image and deploy a new container group from it. During the update process, you are not required to stop the old running container group. You can create a new container group, map the existing route information to it, and then unmap the route from the outdated container group. Your new app is deployed without any downtime.

*Table 4. Optional tools for updating an app*

| Update tools                                                         | Description                                                                                                                                                                                                                                         |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Delivery Pipeline                                                    | Automate your app builds and container deployments to Cloud by using the Delivery Pipeline, which is a Cloud service that is available to you. For more information, see <a href="#">Creating container groups by using the Delivery Pipeline</a> . |
| UrbanCode Deploy                                                     | Automate your app builds and container deployments to Cloud by using UrbanCode Deploy. For more information about purchasing UrbanCode Deploy, see the <a href="#">IBM UrbanCode Deploy</a> product page.                                           |
| Update the app yourself by using the IBM Cloud Container Service CLI | Complete the steps in this task to update the app yourself.                                                                                                                                                                                         |

To update your app yourself by using the IBM Cloud Container Service CLI:

Update your app locally.

Choose to either build your image directly in Cloud or build and test your image locally before you push it to Cloud.

- To build the image directly in Cloud, run the following command.\*

#### Tip

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

```
bx ic build -t registry.DomainName/<my_namespace>/<image_name>:<tag> <source>
```

Example

```
bx ic build -t registry.DomainName/my_namespace/my_image:v1 .
```

#### Note

\*In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk (\*) in this topic.

- To build the image locally and then push the image to Cloud, follow these steps.

If you are using the plug-in for IBM Cloud Container Service, log in again. Do not set the environment variables for option 2, so that `docker` commands will be sent to the Docker engine on your local machine.

```
bx ic init
```

Build the image locally from your Dockerfile.



```
docker build -t <image_name>:<tag> <dockerfile_location>
```

#### Example

```
docker build -t my_ibmliberty_image:v1 .
```

Run a container from the image to test that your new app runs locally by using the following command, where *Port* is the port for HTTP traffic.

```
docker run -d --name <container_name> <image_name>
```

#### Example

```
docker run -d --name my_container my_ibmliberty_image
```

If the app is running correctly, the container ID is displayed in the CLI output. To review the logs for the container, run `docker logs <container_name_or_id>`.

Tag the local image with your private Cloud registry and a new name. Use lowercase alphanumeric characters or underscores (\_) only in the image name. Other symbols, such as hyphens (-) or slashes (/), might prevent the image from being pushed to the image registry.

```
docker tag <current_image_name_or_ID>:<optional_tag> registry.
```

#### Example

```
docker tag my_ibmliberty_image registry.DomainName/<my_namespac
```

Push the image to your private Cloud registry by using the following command.

```
docker push registry.DomainName/<my_namespace>/<image_name>
```

#### Example

```
docker push registry.DomainName/my_namespace/my_ibmliberty_imag
```

## Important

When you push an image to your private Cloud registry, the size reported for the image is smaller than the size of the same image in your local Docker engine. The difference between the sizes does not indicate that an issue occurred when the image was pushed. The compressed size of the image is reported in IBM Cloud Container Service.

Note the route information that is mapped to the running container group.

Run the following command to get the name or ID for the container group.

```
bx ic groups [-q]
```

Run the following command with the group ID or name to retrieve the route.

```
bx ic group-inspect GROUP
```

The route can be found in the "Routes" section. The host and the domain combine to form the full public route URL, such as

`http://mycontainerhost.AppDomainName.`

Create another scalable container group and include the route.

```
bx ic group-create [--anti] [--auto] [-d DOMAIN] [--desired DESIRED] [-e ENVFILE] [--http_monitor_enabled] [--http_monitor_path] [--http_monitor_ip IP_ADDRESS] [-m MEMORY] [--max MAX] [--min MIN] [-n HOST] --name NAME [-p
```

Include the hostname and domain to map the new container group to the route.

For more information about creating container groups, see [Running long-term services as container groups with the command line interface \(CLI\)](#).

Unmap route from the old container group.

```
bx ic route-unmap [-d DOMAIN] [-n HOST] GROUP
```

### Note:

The route unmapping might take up to five minutes to occur.

Optional: Delete the old container group.

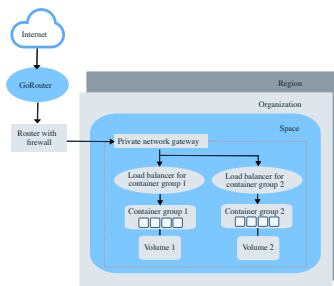
```
bx ic group-remove [-f] GROUP [GROUP]
```

Optional: List all your container groups to verify that the container group was removed.

```
bx ic groups [-q]
```

## Running highly available container groups in the same space

You can create two container groups from the same image in the same Cloud space and map the same route to both groups. With this setup, if a container group or the load balancer for a group goes down, traffic is routed to the other container group.



Before you begin, if you need detailed information about creating container groups, review the information on creating them by using the [CLI](#) or the [Cloud GUI](#).

To create highly available container groups in the same Cloud space:

Create a container group that meets the following requirements.

- The number of instances in the container group must support the expected workload on your app. To determine the number of desired instances, deploy a single container in Cloud that runs your app and perform a load test on this container. If the app can handle, for example, 100 requests per second, but you are expecting an average workload of 300 requests per second, then the desired number of instances for the container group is 3. To make your container group highly available and more resilient to failure, consider including extra instances than the minimum to handle the expected workload. Extra instances can handle the workload in case another instance crashes. To ensure automatic recovery of a failed instance without affecting the workload, include one extra instance. For protection against two simultaneous failures, include two extra instances. This set up is an  $N+2$  pattern, where  $N$  is the number of instances to handle the requests and  $+2$  is an extra two instances.

- Enable auto-recovery for the container group to ensure that crashed containers are recovered automatically.
- Enable anti-affinity. When anti-affinity is enabled, the container instances are spread across separate physical compute nodes, which reduces the likelihood of containers crashing due to hardware failures. Anti-affinity requires using the [CLI](#) and including the `--anti` option in the `bx ic group-create` command. To use the `--anti` option from the CLI, you must have installed the IBM Cloud Container Service plug-in (`bx ic`) version 0.8.934 or later. You might not be able to use this option with larger group sizes because each Cloud region and organization has a limited set of compute nodes available for deployment. If your deployment does not succeed, either reduce the number of container instances in the group or remove the `--anti` option.
- You can store files on a volume and mount the volume to the container or, preferably, store the data off in an external data store, such as a Cloudant® database. Do not include persistent files in the local container file system. Volumes must be created prior to creating a container group.
- Do not create a single container instead of a container group. Single containers are generally to be used for short-term processes and do not have built-in high availability capabilities like container groups.

Example CLI command that includes these suggestions.

```
bx ic group-create --min 3 --max 5 --desired 3 --auto --anti --volume my_
```

Create a second group that meets the following requirements in the same space as the first container group.

- The container group must have a different name, but use the same host name as the previous container group that you created.
- To support the expected workload of the app, use the  $N+2$  pattern as described to determine the number of instances for the container group.
- Enable auto-recovery for the container group to ensure that crashed containers are recovered automatically.
- Enable anti-affinity to spread the container instances across separate physical compute nodes.
- The container group must be mapped to the same route as the first container group you created in the previous step.

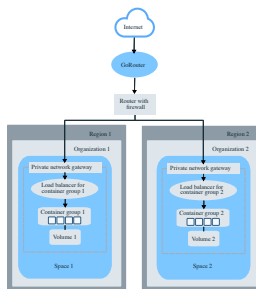
- Do not include persistent files in the container. You can mount the same volume to this container that you mounted to the first container or you can use the same Cloudant database.

## Running highly available container groups in different Cloud regions

If you configure your app's domain name with a Domain Name System (DNS) that provides global load balancing, you can create two container groups from the same image in different Cloud regions that use the same host name in the route. With this setup, you can allow load balancing to occur based on the region the user is in. If the container group, or hardware, or even an entire data center in one region goes down, traffic is routed to the container group that is deployed in another region.

### Important

You must have a custom domain. You cannot use `mybluemix.net`.



Before you begin, if you need detailed information about creating container groups, review the information on creating them by using the [CLI](#) or the [Cloud GUI](#).

To create highly available container groups in different Cloud regions:

Register your custom domain with any certified domain registrar that is capable of global DNS load balancing.

In Cloud, repeat the following steps in all of the regions you want to use.

**Remember:**

If any restrictions exist that prevent you from hosting data in a specific country, do not select a region in that country.

From your account details, in manage organizations, select an organization.

In the **DOMAIN** section, click **ADD DOMAIN** and enter a name for your custom domain.

To allow both HTTPS and HTTP traffic, instead of just HTTP traffic, upload and SSL certificate to Cloud. For more information, see [Securing apps](#).

Click **SAVE**.

Create a container group that meets the following requirements.

- The number of instances in the container group must support the expected workload on your app. To determine the number of desired instances, deploy a single container in Cloud that runs your app and perform a load test on this container. If the app can handle, for example, 100 requests per second, but you are expecting an average workload of 300 requests per second, then the desired number of instances for the container group is 3. To make your container group highly available and more resilient to failure, consider including extra instances than the minimum to handle the expected workload. Extra instances can handle the workload in case another instance crashes. To ensure automatic recovery of a failed instance without affecting the workload, include one extra instance. For protection against two simultaneous failures, include two extra instances. This set up is an  $N+2$  pattern, where  $N$  is the number of instances to handle the requests and  $+2$  is an extra two instances.
- Enable auto-recovery for the container group to ensure that crashed containers are recovered automatically.
- Enable anti-affinity. When anti-affinity is enabled, the container instances are spread across separate physical compute nodes, which reduces the likelihood of containers crashing due to hardware failures. Anti-affinity requires using the [CLI](#) and including the `--anti` option in the `bx ic group-create` command. To use the `--anti` option from the CLI, you must have installed the IBM Cloud Container Service plug-in (`bx ic`) version 0.8.934 or later. You might not be able to use this option with larger group sizes because each Cloud region and organization has a limited set of compute nodes available for deployment. If your deployment does not succeed, either reduce the number of container instances in the group or remove the `--anti` option.
- Do not include persistent files in the container. You can store files in a Cloudant

database that replicates across regions. You can automate the configuration of the database replication by using the [command-line utility](#) that was created by the IBM jStart® team.

- Do not create a single container instead of a container group. Single containers are generally to be used for short-term processes and do not have built-in high availability capabilities like container groups.

Create a second group in a different region. Repeat for any additional regions.

- To support the expected workload of the app, use the  $N+2$  pattern as described to determine the number of instances for the container group.
- Enable auto-recovery for the container group to ensure that crashed containers are recovered automatically.
- Enable anti-affinity to spread the container instances across separate physical compute nodes.
- The container group must be mapped to the same route as the first container group you created in the previous step.
- Do not include persistent files in the container. You can store files in another instance of a Cloudant database.

Configure the DNS to resolve the route to Cloud.

Note the name servers that were assigned to you by the DNS registrar. You must add these servers as NS records in your DNS registrar.

Within your DNS registrar, delegate the domain by providing the name servers.

Configure load balancing and failover for your node.

For a detailed example of setting up highly available CF apps, see [Configure and run a multiregion Cloud application with IBM Cloudant and Dyn](#). The sections that describe how to use Dyn also apply to container groups.

## Connecting to a private container group

Identify the load balancer IP address for a container group in order to access the group without exposing the group to the public.

Every container group is deployed with a built-in [load balancer](#) that balances requests to the

individual containers within a group. Often the container group is exposed publicly with a URL that is mapped to the group. In situations when the group is not public, the container group can be accessed by a private IP address that is automatically assigned to the load balancer. This private IP address also allows apps that communicate over TCP as well as HTTPS to access the container group. In order to determine the private IP address of the load balancer, complete the following steps.

List the container groups that are running in the space. Note the exposed port for the container group, in this case *9080*. If the container group does not have an exposed port, you must create a new container group and expose a port. The container group's load balancer is accessible only on the port that is exposed.

```
bx ic groups
```

| Group ID                             | Name                    | Status    |
|--------------------------------------|-------------------------|-----------|
| cb628ddd-a1e7-4944-9a42-02abc80ea6ee | container_group_example | CREATE_CC |

Inspect the container group. Replace *container\_group\_example* with the name of your container group. Review the **private\_ip\_address** field in the **Loadbalancer** section.

```
bx ic group-inspect container_group_example
```

```
"Loadbalancer": {
 "intermediate_ip_address": "192.0.2.1",
 "private_ip_address": "198.51.100.27"
},
```

**Important:**

The **intermediate\_ip\_address** is used by the GoRouter to securely route traffic from the public network to the container group when the container group is exposed to the public. Do not use the **intermediate\_ip\_address** of the load balancer to access your container group on the private network because it requires additional routing table lookups and can cause network latency.

Identify the load balancer private IP address. An app in the same space can use this private IP address to privately communicate with the container group. You must include the exposed port in the following format to access the load balancer: *198.51.100.27:9080*.

## Removing container groups

To maximize the use of your quota, remove container groups that are not in use occasionally.



## Important

Remove the route from the container group by running one of the following commands before you remove a container group or before you delete a route by using the `cf delete-route` command.

```
bx ic route-unmap -n host -d domain group_name_or_ID
```

Remove a container group by using one of the following methods.

- From the Cloud GUI, select your container group and click **Delete** from the **More actions...** menu.
- ```
bx ic group-remove [-f] GROUP [GROUP]
```

Optional: Verify that the container group was removed by running the following command and confirming that the container group does not appear in the list.

```
bx ic groups
```

Storing persistent data in a volume for single and scalable containers (Deprecated)

A storage volume is used to persist or share data between containers in an organization's space.

Attention

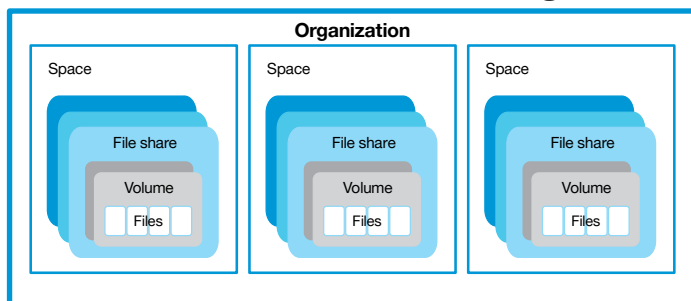
Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

In this page:

- [About volumes](#)
- Creating volumes with the [GUI](#) or the [CLI](#)
- [Adding files](#) to volumes
- [Backing up and restoring](#) volumes
- Adding [non-root user access](#) to volumes
- [Migrating data](#) to a different data center
- [Removing volumes](#)
- [Adding and removing file shares](#)

About volumes

Figure 1. File share and volume structure in an organization.



A container is ephemeral, and you cannot persist application data in it. However, you can use a volume to persist data between container restarts, to share data between containers in a space, and to have access to your application data when your container is not available. If you delete a container, the storage volume is not deleted. The lifecycle of a storage volume is independent from the lifecycle of a container. Since Docker containers in Cloud do not have access to the local host, volumes are mounted on provisioned [file shares](#).

Here are the tasks in the persistent storage lifecycle:

Optional: Create a custom file share that hosts your volumes by using either the [GUI](#) or the [CLI](#). You must have an organization manager role to create a file share. If you are not an organization manager, proceed to creating a volume. One default 20 GB file share is provided for each organization.

Create a volume by using either the [GUI](#) or the [CLI](#).

Create an automatic backup of your volume by using the [ibm-backup-restore image](#).

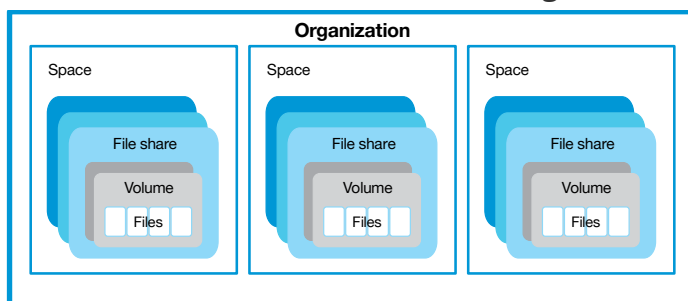
Delete a [volume](#).

Delete a [file share](#).

Creating volumes with the Cloud GUI

A volume is a persistent storage location for the data that an app creates or the files that the app requires to run. You can create a volume for your container from the Cloud GUI.

Figure 2. File share and volume structure in an organization.



A single container is, by design, short-lived. However, you can use a volume to persist data between container restarts, to share data between containers in a space, and to share data between container instances in a group. When you mount a volume in Docker, the volume is

mounted to your local file system. In IBM® Cloud Container Service, the access to the compute host is restricted, so you cannot mount host directories to a container. Instead, organization-scoped volumes are used to persist data between container restarts. Volumes are hosted on isolated [file shares](#) that securely store app data and manage the access and permission to the files. A volume is not affected by any update you make to a container or image. If you delete a container, the storage volume is not deleted.

From the catalog, select **Containers** and choose an image.

Select **Advanced Options** to expand the volume creation fields.

Click **Create a volume**.

In the **Volume name** field, enter a name. Volumes are unique per space, so volume names cannot be reused within a space. The volume name can contain uppercase letters, lowercase letters, numbers, underscores (_), and hyphens (-).

If a file share exists, select an existing file share from the **File share** drop-down menu. If no file share exists, a default file share of 20 GB is created automatically. The organization manager can allocate file shares with specific storage size and IOPS. See [Adding file shares for volumes](#) for more information. There is no size limit for a volume. Volumes can scale to the size of the file share they are hosted on.

Click **Create**. It can take several minutes to create the volume.

The volume is available in your space and can be mounted to your [single container](#) or [container group](#) by selecting it from the drop down menu in **Assign an existing volume** in **Advanced options**. Once you select the volume, assign the file path that the volume is mounted to inside the container, such as `/var/lib/my_volume`. If you require a backup of your container, see [backing up and restoring data](#).

Creating volumes with the command line (CLI)

A volume is a persistent storage location for the data that an app creates or the files that the app requires to run. You can create a volume for your container from the command line.

A single container is, by design, short-lived. However, you can use a volume to persist data between container restarts, to share data between containers in a space, and to share data between container instances in a group. When you mount a volume in Docker, the volume is mounted to your local file system. In IBM Cloud Container Service, the access to the compute

host is restricted, so you cannot mount host directories to a container. Instead, organization-scoped volumes are used to persist data between container restarts. Volumes are hosted on isolated [file shares](#) that securely store app data and manage the access and permission to the files. A volume is not affected by any update you make to a container or image. If you delete a container, the storage volume is not deleted.

Create a volume by running the following command. Volumes can scale to the size of the file share they are hosted on. Specifying a file share is optional. You can view existing file shares by running `bx ic volume-fs`. If no file share exists, a default file share of 20 GB is created automatically. The organization manager can allocate files shares with specific storage size and IOPS. See [Adding file shares for volumes](#) for more information.

```
bx ic volume-create VOLNAME [FSNAME]
```

bx ic volume-create parameters

FSNAME

(Optional) The file share name. If no file share is available or named the volume will be built on the space's default file share.

VOLNAME

(Required) The volume name. The name can contain uppercase letters, lowercase letters, numbers, underscores (_), and hyphens (-).

Optional: Verify that the volume was created correctly by running the following command.

```
bx ic volumes [--table]
```

Next, assign the volume to your [single container](#) or your [container group](#). Volumes are assigned using the `--volume` option in the `run` command and the `group-create` command, respectively. If you require a backup of your container, see [backing up and restoring data](#).

Adding files to volumes with the command line interface (CLI)

After you create a volume, you can mount the volume to a container and add files that are

accessible by any container the volume is mounted to.

Before you begin, create a volume [using the Cloud GUI](#) or [using the CLI](#).

Create a container with a mounted volume.* Replace *my_volume* with the name of your volume, the *container_path* with the mount path of the volume, *my_container* with the name of your container, and *registry.DomainName/my_namespace/my_image* with the image.

```
bx ic run --volume my_volume:/container_path --name my_container registry
```

Tip:

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

Note:

In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk () in this topic.

Copy files from a local directory to the volume by using the `bx ic cp` command.*

In this example, you are copying a *config.txt* file to your volume that is mounted to the */src/usr/tmp* path inside the container. Replace the file with your own file or directory that you are copying and change the container path to point to your volume. In the command, the source file or path is first and the destination path is second. When copying to or from a container, you must include the name of the container, the delimiter ":", and the path to the destination file or folder. The path must be relative to the root directory of the container. See the [bx ic cp](#) documentation for more information.

```
bx ic cp ./config.txt my_container:/src/usr/tmp
```

Confirm that your files were copied to your volume by logging in to your running container with the `bx ic exec` command.*

```
bx ic exec -it my_container bash
```

Navigate to the directory that you mounted your volume to and list the files. Your file is listed in the output. Any container this volume is mounted to has access to this file.

```
# cd container_path
# ls
config.txt
```

Backing up and restoring data with the command line

interface (CLI)

Container volumes are hosted on persistent storage but do not have backups. If you require a backup of your volume data, you can use `bx ic cp` to create a manual backup.

Identify the volume mount path on the running or stopped container that you are backing up.*

```
bx ic inspect my_container
```

The output contains a section similar to this example with the volume name and the container mount point. In this example, the mount point is the folder `/mnt/data`.

```
"Mounts": [  
  {  
    "Destination": "/mnt/data",  
    "Driver": "",  
    "Mode": "rw",  
    "Name": "",  
    "RW": true,  
    "Source": "/vol/fa277ff4-8a64-435b-9b75-0f11d59a3ae4/exam  
  }
```

Note:

In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk () in this topic.

Optional: Stop all running containers that are writing to the volume.* By stopping the containers, you avoid copying data that is being written.

```
bx ic stop my_container
```

Copy the files from the single container to your local computer.* When you want to back up a volume for a container group, it is sufficient to execute this command on one instance of the group as all instances share the same mount point to the volume.

You now have a local back-up of your volume data.


```
bx ic cp my_container:/mnt/data my_local_folder
```

Table 1. The elements of the `bx ic cp` command

Understanding this command	
cp	The copy command.
my_container:/mnt/data	The source path is first. In this case, you are copying your container's data to a local destination, so the container file path is the source. You must include the container name, colon, and the file path from within the container. In this example, you selected the volume mount point from the prior step so that you can copy all included files.
my_local_folder	The destination path is second. The local path can be absolute or relative. In this example, you are copying your container data to the relative path <code>my_local_folder</code> .

Optional: Copy your data from the local folder to a new volume.* This command restores your data to a new container or a new volume. One reason for transferring data to a new volume is that volume is mounted on a newly created file share that has more storage or a higher IOPS than the old file share. See [Adding file shares with the command line interface \(CLI\)](#) for creating a new file share and [Creating volumes with the command line \(CLI\)](#) for creating a new volume.

```
bx ic cp my_local_folder my_new_container:/mnt/data
```

Your data is now copied to your local folder and you have a manual backup. In addition, you copied the data to a volume on a new container.

Adding non-root user access to volumes with the command line interface (CLI)

Non-root users do not have write permission on the volume mount path. To grant write permission, you must edit the Dockerfile of the image to create a directory on the mount path

with the correct permission.

For IBM Cloud Container Service, the user namespace feature is enabled for Docker Engine. User namespaces provide isolation so that the container root user cannot gain access to other containers or the compute host. However, volumes are on NFS file shares that are external to the container compute hosts. The volumes are set up to recognize the root user in the container, but because volumes are external to the container they are not aware of user namespaces.

For some applications, the only user inside a container is the root user. However, many applications specify a non-root user that writes to the container mount path. If you are designing an application with a non-root user that requires write permission to the volume, you must add the following processes to your Dockerfile and entrypoint script:

Create a non-root user.

Temporarily add the user to the root group.

Create a directory in the volume mount path with the correct user permissions.

Create a Dockerfile in a local directory. This example Dockerfile is creating a non-root user named *myguest*.

```
FROM registry.domain_name/ibmnode:latest

# Create group and user with GID & UID 1010.
# In this case your are creating a group and user named myguest.
# The GUID and UID 1010 is unlikely to create a conflict with any existin
# The GUID and UID must be between 0 and 65536. Otherwise, container crea
RUN groupadd --gid 1010 myguest
RUN useradd --uid 1010 --gid 1010 -m --shell /bin/bash myguest

ENV MY_USER=myguest

COPY entrypoint.sh /sbin/entrypoint.sh
RUN chmod 755 /sbin/entrypoint.sh

EXPOSE 22
ENTRYPOINT ["/sbin/entrypoint.sh"]
```

Create the entrypoint script in the same local folder as the Dockerfile. This example entrypoint script is specifying `/mnt/myvol` as the volume mount path.

```
#!/bin/bash
set -e

# This is the mount point for the shared volume.
# Use this mount point with the bx ic run command.
# By default the mount point is owned by the root user.
MOUNTPATH="/mnt/myvol"
MY_USER=${MY_USER:-"myguest"}

# This function creates a subdirectory that is owned by
# the non-root user under the shared volume mount path.
create_data_dir() {
    #Add the non-root user to primary group of root user.
    usermod -aG root $MY_USER

    # Provide read-write-execute permission to the group for the shared vol
    chmod 775 $MOUNTPATH

    # Create a directory under the shared path owned by non-root user mygue
    su -c "mkdir -p ${MOUNTPATH}/mydata" -l $MY_USER
    su -c "chmod 700 ${MOUNTPATH}/mydata" -l $MY_USER
    ls -al ${MOUNTPATH}

    # For security, remove the non-root user from root user group.
    deluser $MY_USER root

    # Change the shared volume mount path back to its original read-write-e
    chmod 755 $MOUNTPATH
    echo "Created Data directory..."
}

create_data_dir
```

```
# This command creates a long-running process for the purpose of this example
tail -F /dev/null
```

Log in to IBM Cloud.

```
bx login [-a api.DomainName] [-sso]
```

The single-sign-on parameter `--sso` is required when you log in with a federated ID. If this option is used, open <https://login.ng.bluemix.net/UAALoginServerWAR/passcode> to obtain a one-time passcode. If you do not include the `--sso` option, complete these substeps.

For **Email**, enter your IBMId for Cloud.

For **Password**, enter the password for the IBMId. Your Cloud organizations and spaces are retrieved.

Enter the number that represents one of your Cloud organizations.

Enter the number that represents one of your existing Cloud spaces.

Initialize the IBM Cloud Container Service CLI.

```
bx ic init
```

Add the image to your registry.* Remember to replace `<my_namespace>` with the namespace to your private images registry. Run `bx ic namespace-get` if you need to find your namespace.

```
bx ic build -t registry.domain_name/<my_namespace>/nonroot .
```

Note:

In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk () in this topic.

Create a volume.

```
bx ic volume-create my_volume
```

Mount the volume and run the container from the *nonroot* image.* The volume mount path `/mnt/myvol` matches the mount path that is specified in the Dockerfile.

```
bx ic run --name my_container --volume my_volume:/mnt/myvol registry.ng.b
```

Confirm that *my_container* is running.* If the container has a BUILDING status, then you

cannot log in to it yet.

```
bx ic ps
```

```
$ bx ic ps
```

CONTAINER ID	IMAGE	COMMAND
d92ee43d-2af	registry.domain_name/<my_namespace>/nonroot:latest	'

Log in to the container after the container is running.*

```
bx ic exec -it my_container /bin/bash
```

View permissions of your volume mount path.

```
ls -al /mnt/myvol/
```

```
root@instance-006ff76b:/# ls -al /mnt/myvol/
total 12
drwxr-xr-x 3 root    root    4096 Jul 13 19:03 .
drwxr-xr-x 3 root    root    4096 Jul 13 19:03 ..
drwx----- 2 myguest myguest 4096 Jul 13 19:03 mydata
```

This output shows that root has read, write, and execute permissions on the volume mount path *mnt/myvol/*, but the non-root *myguest* user has permission to read and write to the *mnt/myvol/mydata* folder. Because of these updated permissions, the non-root user can now write data to the persistent volume.

Migrating volume data to a different data center

If you plan on moving your current Cloud space to a new data center, ensure that your volumes are backed up, so that you can restore and use the data in your new data center.

Attention

When you move your space to a new data center by using the `bx ic reprovision` command, all containers, public IP addresses, and volumes that you have in this space will be permanently removed and not migrated to the new data center. To ensure that data is not lost after moving to the new data center, you must back up all your volumes

and restore the data after your space is reprovisioned in the new data center.

List all the volumes that exist in the Cloud space that you want to reprovision.

```
bx ic volumes
```

Back up every volume that you want to use in the new data center. Choose between the following options.

- Use the [bx ic cp command](#) to back up the volume to your local machine.
- Create a backup container from the [IBM Backup and Restore image](#) to save data in an IBM Object Storage for IBM Cloud instance.

Attention:

If you do not back up your volume and you reprovision your space to the new data center, data is lost and cannot be recovered.

[Remove all volumes in your space.](#)

[Remove all file shares in your space.](#)

Reprovision your space to the new data center. Replace *<data_center>* with the [data center](#) where you want to create your new space.

```
bx ic reprovision <data_center>
```

[Create a file share in the new space.](#)

[Create a volume](#) that you can use to restore the backed up data.

Restore the data into your new volume. Depending on the way that you used to back up your volume, choose between the following options.

- Use the [bx ic cp command](#) to restore the data from your local machine.
- Create a restore container from the [IBM Backup and Restore image](#) to restore data from your IBM Object Storage for IBM Cloud instance.

Removing volumes with the command line interface (CLI)

If you no longer need a volume to store data for your containers, you can remove the volume.

Important

Removing a volume permanently removes any persistent data or files that are stored on the volume. Volumes are not backed up automatically, so [back up your volume](#) before you remove it.

Identify the volume name.

```
bx ic volumes [--table]
```

Inspect your volume to review which containers the volume is mounted on.

```
bx ic volume-inspect example_volume
```

The output contains a section that is named **mounted_on_containers** that lists containers that the volume is mounted on.

```
"mounted_on_containers": [  
  "example_container"  
],
```

If the volume is mounted to a container, remove any confidential data before you delete the volume.

Optional: Inspect your container to find the mount path that the volume is mounted on.

```
bx ic inspect example_container
```

You can find the mount path in the **Volumes** section of your CLI response.

```
"Volumes": {  
  "/mount_path": "/vol/0e65c452-62413-46f2-8b22-ff06ab74ac21/"
```

Log in to the container.

```
bx ic exec -it example_container bash
```

Navigate to your mount path.

```
cd /mount_path
```

Remove all confidential data from your volume.

```
rm -rf *
```

Verify that all data is removed.

```
ls
```

If no directories or files are displayed, data is successfully removed from the volume mount path.

Repeat steps a-e for all containers that are mounted to the volume and that contain confidential data in their volume mount path.

Stop all the containers that are mounted to your volume prior to removing the volume.*

```
bx ic stop example_container
```

Note:

In this command, you can replace `bx ic` with `docker` when you [logged in to IBM Cloud Container Service](#) and set your environment variables to use native Docker commands. You can use native Docker commands in all steps that are marked with an asterisk () in this topic.

If the container is no longer needed, remove it.*

```
bx ic rm example_container
```

When all of the containers with the volume mounted are stopped, remove the volume.

```
bx ic volume-remove VOLNAME
```

bx ic volume-remove parameters

VOLNAME

(Required) The volume name.

Optional: Verify that your volume is removed.

```
bx ic volumes [--table]
```

Adding file shares for volumes with the Cloud GUI

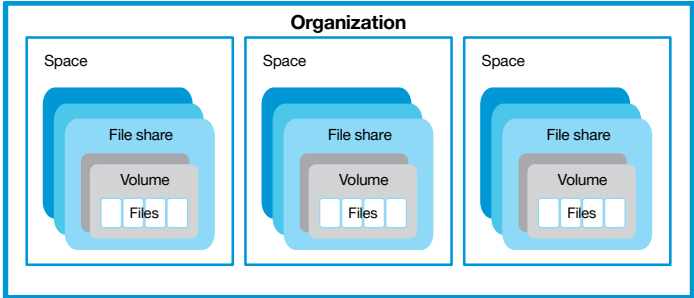
A file share is a persistent NFS-based (Network File System) storage system that hosts Docker volumes in a Cloud space, and allows a user to store and access container and app-related files. To store files in a file share, you must create a container volume and save the data into this volume.

Note

If you have already reached your maximum quota limits, you can request to add extra quota by opening a [Cloud support ticket](#).

Since containers in Cloud do not have access to the local host, volumes are mounted on provisioned NFS file shares. The volumes on a file share are similar to directories on a host computer. Both file shares and volumes are unique per space, so a container must be created in the same space to access the persistent data on the volume. For more on setting up organizations and spaces, see [Managing IBM Cloud Public](#). Upon the first creation of a volume in an organization, a file share of 20 GB and 4 IOPS is provisioned automatically to host the volume. To meet the performance needs for the data, the organization manager can create new [file shares sizes](#) from 20 GB to 12 TB and at IOPS per GB of 0.25, 2 or 4.

Figure 3. File share and volume structure in an organization.



- From your account details, in manage organizations, select an organization.
- In the quota section, view the details for containers.
- Create a file share. A row is created at the end of the table.
- Select an existing space for the file share to be created in.
- Enter a name for the file share. The name can contain uppercase letters, lowercase letters,

numbers, underscores (_), and hyphens (-).

Determine the capacity your file share needs. The file share size and IOPS per GB (input/output operations per second) multiply together to create the total IOPS. The table illustrates the relationship between file share size and IOPS.

Table 2. Examples of file share sizes and IOPS per GB

File share sizes	IOPS per GB	Total IOPS
20 GB	4	80
100 GB	4	400
1,000 GB	4	4,000
4,000 GB	4	16,000

Apps with higher workloads require a greater total IOPS, so consider the apps that write to your volumes when you choose the file share configuration.

For the **Size**, select a value in gigabytes.

For the **IOPS**, select a value in IOPS per gigabyte. Based on the inputs for **Size** and **IOPS**, a monthly price is calculated for you. Before you continue, you can adjust the values for **Size** and **IOPS** to change the monthly price calculation.

Click **Save**.

It can take several minutes for the file share creation to complete. After the file share is finished building, the status icon displays a check mark. You now have a file share available in the specified space in your organization. You can create more file shares to host more volumes or you can host multiple volumes on one file share. Every volume shares the resources of the file share it is hosted on so hosting multiple volumes on one file share can reduce performance.

Next, if you are an organization manager you can view your organization's file share billing.

Click on the user avatar.

Click on **Account**.

Expand your organization by clicking the + button.

Select **Usage Dashboard** and review the usage and billing details.

For more information, go to [Managing your account](#).

Adding file shares with the command line interface (CLI)

Create a file share by using the `bx ic volume-fs-create` command.

Optional: List the valid file share sizes by running the following command.

```
bx ic volume-fs-flavors
```

Create a file share by running the following command. To view calculated costs of the file share before creating the file share, see [Adding file shares for volumes with the Cloud GUI](#)

```
bx ic volume-fs-create FSNAME FSSIZE FSIOPS
```

bx ic volume-fs-create parameters

FSIOPS

(Required) The file share IOPS. Valid values are 0.25, 2 or 4 IOPS per GB.

FSNAME

(Required) The file share name. The name can contain uppercase letters, lowercase letters, numbers, underscores (_), and hyphens (-).

FSSIZE

(Required) The file share system size in GB. Use `bx ic volume-fs-flavors` to list all valid sizes

Optional: Verify that the file share was created correctly by running the following command.

```
bx ic volume-fs
```

It can take several minutes for the file share creation to complete. After the file share is finished building, the status icon displays a check mark. You now have a file share available in the specified space in your organization. You can create more file shares to host more volumes or you can host multiple volumes on one file share. Every volume shares the resources of the file share it is hosted on so hosting multiple volumes on one file share can reduce performance. Next, if you are an organization manager you can view your organization's file share billing.

Click on the user avatar.

Click on **Account**.

Expand your organization by clicking the **+** button.

Select **Usage Dashboard** and review the usage and billing details.

For more information, go to [Managing your account](#).

Removing file shares

An organization manager can remove existing file shares and view which volumes are mounted on a file share.

The organization manager can review file shares by using both the Cloud GUI and the CLI to see which volumes are mounted on them and to remove the file shares. Organization managers can opt to remove file shares that are not currently being used in development or in production.

Note

File shares cannot be deleted at the end of the month due to the bill processing cycle. If you receive an error message at the end of the month when deleting a file share, wait until the first day of the upcoming month to delete the file share.

Using the Cloud GUI

From your account details, in manage organizations, select an organization.

In the quota section, view the details for containers.

In the File Storage table, you can find an **ACTION** column and a **NAME** column.

Click the name of the file share to view the volumes attached to it.

Click **Delete** to delete the file share. You must first remove any attached volumes in order to delete a file share. Once a file share is deleted, it is permanently removed from your account. To remove a volume, see [Removing volumes with the command line interface \(CLI\)](#).

Using the command line

To remove a file share by using the CLI, issue the following commands.

Identify the file share name.

```
bx ic volume-fs
```

Inspect the file share to determine whether any volumes exist on it.

```
bx ic volume-fs-inspect FSNAME
```

If needed, remove an existing volume.

```
bx ic volume-remove VOLNAME
```

Remove an existing file share.

```
bx ic volume-fs-remove FSNAME
```

List the existing file shares to confirm that the file share is removed.

```
bx ic volume-fs
```

Integrating services with single and scalable containers (Deprecated)

You can use various services in the catalog with IBM® Cloud® Container Service. Many of the Cloud services help you to bring extra capabilities to the apps that you deploy in containers. Additionally, some Cloud services can be used to help you manage your containers.

Attention

Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

In this page:

- [Data analytics and Storage services](#)
- [DevOps services](#)
- [Integrate services](#)
- [Network services](#)
- [Binding a service from the Cloud GUI](#)
- [Binding a service to a container from the CLI](#)
- [Configuring alerts for apps that are deployed in containers by using Alert Notification](#)
- [Automating your container deployments by using the Delivery Pipeline in the Continuous Delivery service](#)
- [Tutorial: Deploying a Cloud Foundry Node.js app in a container](#)

Data analytics and Storage services

Object Storage

- Store persistent data that is replicated across multiple storage nodes to back it up.

For general information about the Object Storage service, see [Getting started with the Object Storage service](#).

DevOps services

Alert Notification

Set notification policies to filter only the alerts that you are interested in getting from your application. You can refine the policy filters as needed to specify individuals or groups as recipients of alerts, as well as preferred notification methods and frequency.

For more information about using containers with Alert Notification, see [Configuring alerts for apps that are deployed in containers by using Alert Notification](#).

For general information about the Alert Notification service, see [Getting started with Alert Notification](#).

Delivery Pipeline

Automate your app builds and container deployments to Cloud by using Delivery Pipeline, which is part of IBM Cloud Continuous Delivery.

For more information about using containers with the pipeline, see [Automating your container deployments by using the Delivery Pipeline in the Continuous Delivery service](#).

For general information about the Delivery Pipeline service, see [Getting started with Delivery Pipeline](#).

UrbanCode Deploy

Manage your releases and automate container deployments with IBM UrbanCode Deploy. This service provides you with tools that can be used to have visibility into your deployments across every environment. You can use this service as an alternative to the Delivery Pipeline service.

For purchasing information, see the product page for [IBM UrbanCode Deploy](#).

For general information about UrbanCode Deploy, see the [IBM Knowledge Center](#).

Integrate services

Istio

Manage your microservices in IBM Cloud Container Service for an application or service. For information or to get involved in the Istio open source project, see <https://istio.io/>.

Secure Gateway

Securely connect Cloud applications to remote locations on-premises or in the cloud.

For general information about the Secure Gateway service, see [Getting started with the Secure Gateway service](#).

Network services

VPN

Allow containers to communicate to one another between a Cloud space and an intranet server.

For general information about the IBM VPN service, see [Getting started with the IBM VPN service](#).

Related links

[Docker Tooling for Eclipse](#)

Binding a service from the Cloud GUI

IBM Cloud has a list of services and manages them on behalf of app developers. To add a Cloud service for your container to use, you must request an instance of this service and bind the service to the container.

Before you begin, add the service to your Cloud space. For more information, see [Requesting a new service instance](#).

Restriction

Some Cloud services do not support the binding directly to a container and lead to a failure during container creation. In this case, bind the Cloud service to a Cloud Foundry app first, and then bind the app to the container. The app acts as a bridge between the service and the container. For more information, see [Cloud service does not bind directly to a container](#).

In the Cloud Catalog, click **Containers**.

Click a container image.

Expand **Advanced Options** and for **Select a service**, select an available Cloud service.

Click **Add**. The Cloud service is displayed in the **Already bound to your container** field.

Complete the other fields, and then click **CREATE**.

When the container is created, verify that the VCAP_SERVICES information is displayed. In the Overview tab for the container, locate the bound service and click **View Credentials**. The information that is displayed is the VCAP_SERVICES.

Binding a service to a container from the CLI

IBM Cloud has a list of services and manages them on behalf of app developers. To add a Cloud service for your container to use, you must request an instance of this service and bind the service to the container.

Before you begin, add the service to your Cloud space. For more information, see [Requesting a new service instance](#).

Restriction

Some Cloud services do not support the binding directly to a container and lead to a failure during container creation. In this case, bind the Cloud service to a Cloud Foundry app first, and then bind the app to the container. The app acts as a bridge between the service and the container. For more information, see [Cloud service does not bind directly to a container](#).

Use one of the following commands to bind the Cloud service instance to a new container.

Tip:

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

- o If you are creating a single container, use the following command.

```
bx ic run --name container_name -e "CCS_BIND_SRV=service_instance1,service_instance2"
```

- o If you are creating a container group, use the following command.

```
bx ic group-create --name container_group_name -e "CCS_BIND_SRV=service_instance1,service_instance2"
```

- o If you are binding a service to container group that is already running, use the following command. This command is available for container groups only. Only one service can be bound at a time.

```
bx ic service-bind container_group_name service_instance
```

Use one of the following commands to verify that the service instance is bound to the container.

- o For a single container, use the following command.

```
bx ic inspect CONTAINER
```

- o For a container group, use the following command.

```
bx ic group-inspect GROUP
```

Verify that the VCAP_SERVICES information for the service is displayed.

```
{  
  "VCAP_SERVICES": {
```

```

"<service_name>": [
  {
    "credentials": {
      "hostname": "<hostname_id>",
      "jdbcUrl": "<jdbc_URL>",
      "name": "<encoded_name>",
      "password": "<encoded_password>",
      "port": "<port_number>",
      "uri": "<service_URI>",
      "username": "<encoded_username>"
    },
    "label": "<service_name>",
    "name": "<service_instance>",
    "plan": "<plan_name>",
    "tags": [
      "<tag1>",
      "<tag2>",
      "<tag3>"
    ]
  }
]
}

```

Configuring alerts for apps that are deployed in containers by using Alert Notification

Configure your app to send notifications when a certain condition or criteria is met and use the Alert Notification service to manage your alert notifications. With the Alert Notification service you can create custom notification policies that filters the alerts that you are interested in. You can refine the policy filters as needed to specify individuals or groups as recipients of alerts, as well as preferred notification methods and frequency.

Before you begin, review the [Requesting a new service instance](#) topic and create an instance of the Alert Notification service in your Cloud space.

The following steps showcase how to create a container that runs a sample Alert Notification node app. You can send test alerts to the Alert Notification service and review them in the Alert

Viewer tool.

Download and clone the sample [Alert Notification app](#) to your local machine.

```
git clone https://github.com/IBM-Cloud/alert-notification-helloworld.git
```

Note:

When you clone the Git project, a new folder named `alert-notification-helloworld` is created on your local machine.

Build a container image that includes the sample Alert Notification app and push it to your private Cloud images registry.

[Log into the IBM Cloud Container Service CLI.](#)

Navigate to the `alert-notification-helloworld` directory.

Create your container image and push it to your private Cloud images registry. Run `bx ic namespace-get` to retrieve your namespace information and replace `<my_namespace>` with your namespace.

```
bx ic build -t registry.DomainName/<my_namespace>/alert .
```

Verify that your container image is successfully created and pushed.

```
bx ic images
```

Create a container from the image and bind the Alert Notification service to it. Replace `<my_namespace>` with your namespace information and `<my_alert>` with the name of the Alert Notification service instance that you created in the beginning.

```
bx ic run --name alertcontainer -p 8080 -e "CCS_BIND_SRV=<my_alert>" regi
```

Verify that your container is running.

```
bx ic ps
```

Bind a public IP address to your container.

List the public IP addresses that are available in your space and note the IP address you want to use.

```
bx ic ips
```

Tip:

If no public IP addresses are available in your space, run `bx ic ip-request` to request a new IP address. You can also unbind an used IP address from a container

by running the `bx ic unbind <IP_ADDRESS> <CONTAINER>` command.

Bind the public IP address to your container. Replace `<IP_ADDRESS>` with the IP address you noted in the previous step.

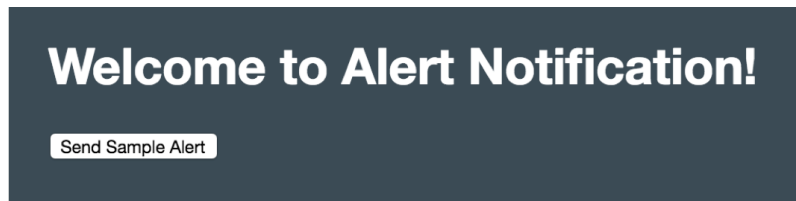
```
bx ic ip-bind <IP_ADDRESS> alertcontainer
```

Create a test alert.

Open your web browser.

Enter the IP address and port in your web browser as follows:

`<IP_ADDRESS>:8080`. The welcome page of your Alert Notification web app opens.



Click on the **Send Sample Alert** button to send a test alert to the Alert Viewer.

Open the Alert Viewer and review your test alert.

From the Cloud GUI, select your running container instance.

Click on the Alert Notification service under your **Bound services**.

Click on **Launch** to open the Alert Viewer.

Review your test alert.

Explore the capabilities of the Alert Notification service. Review the Alert Notification documentation to find more information about how to [create notification policies](#) and [manage your alerts](#).

Automating your container deployments by using the Delivery Pipeline in the Continuous Delivery service

Automate your app builds and container deployments to Cloud by using the Delivery Pipeline, which is part of IBM Cloud Continuous Delivery.

The Delivery Pipeline supports:

Building Docker images

Deploying images in containers to Cloud

Quick start guide for deploying containers by using the Delivery Pipeline:

Create a Git repository that contains your code.

Include a `Dockerfile` in the root directory of your repo.

From the menu, click **Services > DevOps**, and then click **Pipelines**.

Click **Create Pipeline > Delivery Pipeline**.

Click **Create**.

Add your Git repo to the toolchain.

Click **Add a Tool**.

Click the type of Git service that hosts your repo.

Provide the configuration details for your repo.

Click **Create Integration**.

From the toolchain's Overview page, click **Delivery Pipeline**.

Create a build stage that includes a build job.

Click **Add Stage**.

On the **INPUT** tab, for the **Input type**, select **Git Repository**.

On the **JOBS** tab, add a build job. For the **Builder type**, select **IBM Containers on Cloud**.

Create a deployment stage that includes a deployment job.

On the **INPUT** tab, for the **Input type**, select **Build Artifacts**.

On the **JOBS** tab, for the **Deployer type**, select **IBM Container Service**.

Run a deployment by either pushing a code change or by clicking the Run Stage icon on the build stage of the Delivery Pipeline.

Need more information? Look at the following topics.

[Creating a single container by using the Delivery Pipeline](#)

[Creating container groups by using the Delivery Pipeline](#)

Creating a single container by using the Delivery Pipeline

Automate the building and deploying of your single containers with the Delivery Pipeline.

Creating build stages for containers by using the Delivery Pipeline

IBM Cloud Container Service includes a build service that takes in source code and then builds and publishes a Docker image to the container registry. Delivery Pipeline, which is part of IBM Cloud Continuous Delivery, provides a build job that uses the container build service. This build job takes the source code from a repository, builds a Docker image from that code, versions (tags) that image to match the build number, then pushes the resulting image to the organization's IBM Cloud Container Service registry on Cloud. Then, members of the organization can deploy the resulting image to spaces within the organization.

Before you begin, consider the following information.

To learn more about builds, stages, and basic pipeline functionality, see the [About Delivery Pipeline](#) topic in the Continuous Delivery documentation.

Identify an existing image from your organization's private images registry to use in your container by checking the Cloud Catalog.

If you know that you want to bind a service to the container, create an app to use as a bridge. For more information, see [Binding a service to a container by using the Delivery Pipeline](#).

Build a Docker image from Delivery Pipeline.

Create a Dockerfile in the root directory of the Git repository that is associated with your toolchain and commit it to your repository.

From the toolchain's Overview page, click **Delivery Pipeline**.

Create a build stage.

Click **Add Stage**.

Name the stage. For example, *Build*.

For the **Input type**, select **Git Repository**.

For the other fields, verify the values and update, if necessary.

Click the **JOBS** tab.

In the build stage, create a build job.

In the **JOBS** tab, click **ADD JOB**.

Select the **Build** job type.

For the **Builder type**, select **IBM Containers on Cloud**.

If necessary, update the Cloud target, organization, and space.

For **Use cached layers**, clear the field when you must force a refresh of layers that dynamically pulls content from other sources.

For example, if the Dockerfile includes an `apt-get install` or `wget` command, ensure that these dynamic dependencies are gathered each build by clearing the **Use cached layers** check box.

Note:

When the **Use cached layers** field is cleared, build times are longer because cached layers are not used when the image is building.

In the **Image Name** field, enter a name for the image to build. To meet Docker naming conventions, use lowercase letters, numbers, and hyphens (-) only. When the image is built, it is tagged with the image name and a build number. For example, if you enter the image name `myappimage`, the resulting image for build number 4 is called `myappimage:4`.

If your Dockerfile is not located in the root directory of the project, in the Build Script section, add the path to the Dockerfile in the following locations.

Before the Run unit tests section, add the folder name with a change directory command.

```
cd <folder>
```

Wherever the `${WORKSPACE}` variable is displayed, append the folder name. The `${WORKSPACE}` variable automatically detects the root directory of the project for you.

Example


```
BUILD_COMMAND="build --pull --tag ${FULL_REPOSITORY_NAME} ${FULL_REPOSITORY_NAME}
```

Click **SAVE**.

Click the **Run Stage** icon to verify that the stage builds without errors. If the build fails, review the logs and correct the errors. If the stage passes, check your organization's image registry and verify the image was built.

Creating deployment stages for containers with the Delivery Pipeline

You can create a deployment stage for your container by using the Delivery Pipeline.

Before you begin:

[Create a build stage.](#)

[Review the default deployment script for single containers.](#) You can fork the default script and edit it or create your own script to use instead.

[Review a list of available environment variables that are provided by the Delivery Pipeline service.](#)

From the toolchain's Overview page, click **Delivery Pipeline**.

Create a deployment stage.

Click **ADD STAGE**.

Name the stage. For example, *Deploy*.

For the input type, select **Build Artifacts**.

In the stage, create a deployment job.

On the **JOBS** tab, click **ADD JOB**.

For the job type, select **Deploy**.

For the deployer type, select **IBM Container service on Cloud**.

For the organization and space, verify the values and update, if necessary.

In the **Deployment strategy** field, enter one of the following values.

- **red_black** - Deploys a new version of the app. If the deployment is successful,

the previous deployment of the app is removed and the IP address or route is mapped to the new version.

- **clean** - Removes all previous deployments of the application.

Tip:

The `CONCURRENT_VERSIONS` environment property sets the number of versions of the app deployments to keep active. The default value is one version. For example, if `CONCURRENT_VERSIONS` is set to 2, and you are using an IP address, the most recent deployment is bound to the IP address, but the previous version is still deployed for a quick standby.

To always remove all deployments of the app, in the **ENVIRONMENT PROPERTIES** tab, you can set `CONCURRENT_VERSIONS` to 0.

In the **Name** field, enter the name of the container to deploy.

For the port, enter the published port for the container. Separate multiple ports with a comma (,).

Tip:

If no port is specified and you are using the default deployment script, port 80 is automatically published. To make a container private with the default deployment script, complete the following steps.

For deployment strategy, enter `simple`.

Leave the **Port** field blank.

Because simple deployments do not clean up previous deployments, you can create another deployment stage to clean them up. Set the **Deployment strategy** in the second deployment stage to `clean`.

If you have any optional deployment arguments, enter them. For example, if your app requires an environment variable for the location of your back-end database, you might include this line:

```
--env MY_DB_HOSTNAME_PROPERTY=mydbhostname
```

If you created your own deployment script, edit the **Deployer script** section to use it.

Copy the following line and update it to use your repository.

```
git_retry clone https://github.com/0sthane/deployscripts.git deplo
```

Edit the following line to use your script.

```
/bin/bash deployscripts/deploycontainer.sh
```

Click **SAVE**.

Creating container groups by using the Delivery Pipeline

Automate the building and deploying of your container groups by using the Delivery Pipeline.

Creating build stages for container groups by using the Delivery Pipeline

IBM Cloud Container Service includes a build service that takes in source code and then builds and publishes a Docker image to the container registry. Delivery Pipeline, which is part of IBM Cloud Continuous Delivery, provides a build job that uses the container build service. This build job takes the source code from a repository, builds a Docker image from that code, versions (tags) that image to match the build number, then pushes the resulting image to the organization's IBM Cloud Container Service registry on Cloud. Then, members of the organization can deploy the resulting image to spaces within the organization.

Before you begin, consider the following information.

To learn more about builds, stages, and basic pipeline functionality, see the [About Delivery Pipeline](#) topic in the Continuous Delivery documentation.

Identify an existing image from your organization's private images registry to use in your container by checking the Cloud Catalog.

If you know that you want to bind a service to the container, create an app to use as a bridge. For more information, see [Binding a service to a container by using the Delivery Pipeline](#).

Build a Docker image from Delivery Pipeline.

Create a Dockerfile in the root directory of the Git repository that is associated with your toolchain and commit it to your repository.

From the toolchain's Overview page, click **Delivery Pipeline**.

Create a build stage.

Click **Add Stage**.

Name the stage. For example, *Build*.

For the **Input type**, select **Git Repository**.

For the other fields, verify the values and update, if necessary.

Click the **JOBS** tab.

In the build stage, create a build job.

In the **JOBS** tab, click **ADD JOB**.

Select the **Build** job type.

For the **Builder type**, select **IBM Containers on Cloud**.

If necessary, update the Cloud target, organization, and space.

For **Use cached layers**, clear the field when you must force a refresh of layers that dynamically pulls content from other sources.

For example, if the Dockerfile includes an `apt-get install` or `wget` command, ensure that these dynamic dependencies are gathered each build by clearing the **Use cached layers** check box.

Note:

When the **Use cached layers** field is cleared, build times are longer because cached layers are not used when the image is building.

In the **Image Name** field, enter a name for the image to build. To meet Docker naming conventions, use lowercase letters, numbers, and hyphens (-) only. When the image is built, it is tagged with the image name and a build number. For example, if you enter the image name `myappimage`, the resulting image for build number 4 is called `myappimage:4`.

If your Dockerfile is not located in the root directory of the project, in the Build Script section, add the path to the Dockerfile in the following locations.

Before the Run unit tests section, add the folder name with a change directory command.

```
cd <folder>
```

Wherever the `${WORKSPACE}` variable is displayed, append the folder name. The `${WORKSPACE}` variable automatically detects the root directory of the project for you.

Example

```
BUILD_COMMAND="build --pull --tag ${FULL_REPOSITORY_NAME} ${
```

Click **SAVE**.

Click the **Run Stage** icon to verify that the stage builds without errors. If the build fails, review the logs and correct the errors. If the stage passes, check your organization's image registry and verify the image was built.

(Optional) Checking image security as part of the pipeline

Find security or configuration issues in your image before the container is deployed by using Vulnerability Advisor.

Common pipeline setup that includes this scan:

- Build stage to build the source files

- Processing stage, which includes the following jobs:

- A build job to run a container build
- A test job to run Vulnerability Advisor on an image

- Deploy stage to deploy the container

Tip

This task outlines how to create a test job for Vulnerability Advisor in a different stage than the job that builds the container image. Alternatively, you can use the same stage. If you decide to create a separate stage, you are not required to set the environment variable.

Create the Vulnerability Advisor scan.

Create a processing stage.

Click **Add Stage**.

Name the stage. For example, *Processing*.

For the input type, select **Build Artifacts**.

For the stage and job, verify the values and update, if necessary.

Click the **JOBS** tab.

In the processing stage, add a test job to run the code scan.

On the **JOBS** tab, click **ADD JOB**.

For the job type, select **Test**.

For the **Tester type**, select **IBM Vulnerability Advisor**.

If necessary, update the Cloud target, organization, and space.

For **Minutes to wait for analysis to complete**, enter a value 0 - 59 minutes. The default value is 5 minutes. A URL to the dashboard is provided in the console logs at the end of the job.

If the scan is not complete before this time ends, the job fails, but the scan analysis continues to run and can be viewed on the dashboard. After the scan is complete, if you rerun the job, the scan request is not resubmitted and the pipeline job can complete successfully. Alternatively, you can configure the pipeline not to be blocked on a successful scan result, which is described in the next step.

To determine the behavior on timeout, select or clear the **Stop running this stage if this job fails** check box.

You can configure the job to block both later jobs in the stage and later stages from running if this job fails, or you can configure the stage to continue without blocking later jobs and stages. The state of the job is determined by the existence of major security issues.

Major security issues cause the job to fail. Depending on the selection for **Stop running this stage if this job fails**, a failed job either stops the stage and pipeline from progressing or allows them to continue. If you know that the report includes many issues to process, you might configure the stage to continue because the scan can take a long time. In this scenario, you might not want the rest of your jobs and stages to stop only because the scan is taking too long.

If the job is in the same stage as the build job, set an environment variable for the repository name.

Click the **ENVIRONMENT PROPERTIES** tab.

Click **ADD PROPERTY** and select **Text Property**.

In the **Name** field, enter FULL_REPOSITORY_NAME

In the **Value** field, do not enter anything. The image name that is specified in the container build job is passed through this environment variable.

Click **SAVE**.

Click the **Run Stage** icon to verify that the stage builds without errors.

To view the results of a Vulnerability Advisor scan, after the job completes, click **View results and history**. On the **LOGS** tab, look for the following section:

```
image registry.DomainName/namespace/image vulnerability results found
```

(Optional) Cleaning up images with the Delivery Pipeline

The IBM Cloud Container Service build job cleans up unused image versions automatically. Images that are in use are not removed. You can specify a number of image versions to keep or you can disable the image cleanup.

To change the number of image versions that are kept for a project:

In the pipeline, open the configuration for the build stage.

On the **ENVIRONMENT PROPERTIES** tab, click **ADD PROPERTY**.

Select **Text property**.

In the **Name** field, enter the IMAGE_LIMIT command.

In the **Value** field, enter the number of image versions for this project to keep in the registry. By default, the five most recent image versions for a project are kept. To disable the image cleanup and keep all of the image versions, set the value to -1.

Click **SAVE**.

(Optional) Binding a service to a container by using the Delivery Pipeline

Services cannot be directly bound to a container. As an alternative, you can bind the service to an app, and then bind that app to the container by configuring an environment property in a

pipeline stage. The app serves as a bridge between the service and the container.

Cloud provides a set of environment variables that define the names and details of the services that are bound to an app, such as ClearDB MySQL database or Question and Answer. These environment variables are called VCAP_SERVICES. When a container is bound to a service, the VCAP_SERVICES are passed into the container as an environment variable through an app that acts as a bridge between the container and the service. Then, the container can look up information from the service, such as the endpoint, user, and password.

Before you configure the environment property, identify an existing app that has services that are bound to it already or create an app by using the Cloud GUI or the command line. For more information, see [Cloud service does not bind directly to a container](#).

Bind the bridge app to the container.

From the toolchain's overview page, click **Delivery Pipeline**.

Open a pipeline stage and click the **ENVIRONMENT PROPERTIES** tab.

Click **ADD PROPERTY** and select **Text property**.

In the **Name** field, enter the command BIND_TO.

In the **Value** field, enter the name of the bridge app, such as *pipeline_bridge_app*.

Click **SAVE**.

Test the configuration by running your pipeline.

Tip

After the bridge app is bound to a container, do not delete the bridge app. If you delete a bound bridge app, the container hangs. Delete the container, and any other containers that are using the bridge app, before you delete the bridge app.

Creating deployment stages for container groups by using Delivery Pipeline

You can create a deployment stage for your container group by using the Delivery Pipeline.

Before you begin:

Create a build stage.

Review the default deployment script for container groups. You can fork the default script and edit it or create your own script to use instead.

Review a list of available environment variables that are provided by the Delivery Pipeline service.

Create a deployment stage.

Click **ADD STAGE**.

Name the stage. For example, *Deploy*.

For the input type, select **Build Artifacts**.

On the stage, create a deployment job.

On the **JOBS** tab, click **ADD JOB**.

For the job type, select **Deploy**.

For the deployer type, select **IBM Container service on Cloud**.

Select the organization and space.

For the deployment strategy, enter one of the following values.

- **red_black** - Deploys a new version of the app. If the deployment is successful, the previous deployment of the app is removed and the IP address or route is mapped to the new version.
- **clean** - Removes all previous deployments of the application.

For the name, enter the name of the container to deploy.

For the port, enter the published port for the container. For container groups, you can only specify one port.

Tip:

If no port is specified and you are using the default deployment script, port 80 is automatically published. To make a container private with the default deployment script, complete the following steps.

For deployment strategy, enter `simple`.

Leave the **Port** field blank.

Because simple deployments do not clean up previous deployments, you

can create another deployment stage to clean them up. Set the **Deployment strategy** in the second deployment stage to clean.

If you have any optional deployment arguments, enter them. For example, if your app requires an environment variable for the location of your back end database, you might include this line:

```
--env MY_DB_HOSTNAME_PROPERTY=mydbhostname
```

Set up a container group deployment instead of a single container deployment. In the "Deployer script" section, remove the # to uncomment the container group script:

```
/bin/bash deployscripts/deploygroup.sh
```

Add a # to comment out the single container script:

```
#!/bin/bash deployscripts/deploycontainer.sh
```

If you created your own deployment script, edit the **Deployer script** section to use it.

Copy the following line and update it to use your repository.

```
git_retry clone https://github.com/Osthane/deployscripts.git deplo
```

Edit the following line to use your script.

```
/bin/bash deployscripts/deploygroup.sh
```

Optional: To configure a basic container group, set each of the environment variables in the table. These properties allow you to set some of the options that you might set when you are deploying containers from the command line or the Cloud user interface. For other properties, see the comments in the Deployer script section of the user interface.

Click the **ENVIRONMENT PROPERTIES** tab and click **ADD PROPERTY**.

Select **Text property**.

Enter the name and value of an environment property.

Property	Description
AUTO_RECOVERY	true or false When automatic recovery is enabled for a container group and the group has been running for 10 minutes, the group's load balancer starts to regularly check the health of each container instance in the group via HTTP requests. If a container instance does not

Property	Description
	<p>respond within 100 seconds, it is marked as inactive. Inactive container instances are removed from the group and re-created by auto-recovery. Auto-recovery attempts to recover container instances in a group 3 times. After the third attempt, auto-recovery does not recover any container instances in the group for 60 minutes. After 60 minutes, the auto-recovery process starts again.</p>
CONCURRENT_VERSIONS	<p>A number</p> <p>The CONCURRENT_VERSIONS environment property sets the number of versions of the app deployments to keep active. The default value is one version. For example, if CONCURRENT_VERSIONS is set to 2 and you are deploying a container group, two groups are mapped to the route and receive traffic.</p> <p>To always remove all deployments of the app, in the ENVIRONMENT PROPERTIES tab, you can set CONCURRENT_VERSIONS to 0.</p>
DESIRED_INSTANCES	<p>A number</p> <p>Example: 4</p> <p>The number of container group instances that your app requires depends on the expected workload and the time in which a request can be processed by the app. For example, multi-tasking or network connectivity affects how many instances might be required for your app.</p> <p>To make your container group highly available and more resilient to failure, consider including extra instances than the minimum to handle the expected workload. Extra instances can handle the workload in</p>

Property	Description
	<p>case another instance crashes. To ensure automatic recovery of a failed instance without affecting the workload, include one extra instance. For protection against two simultaneous failures, include two extra instances. This set up is an $N+2$ pattern, where N is the number of instances to handle the requests and $+2$ is an extra two instances.</p>
ROUTE_DOMAIN	<p>A string or none Example: <code>mycustomdomain.com</code></p> <p>The default system domain is <code>AppDomainName</code> and already provides a SSL certificate, so you can access your container groups with HTTPS without any additional configuration. To use a custom domain, you must register the custom domain on a public DNS server, configure the custom domain in Cloud, and then map the custom domain to the Cloud system domain on the public DNS server.</p> <p>If a domain is not specified, <code>AppDomainName</code> is used. If none is specified, no domain is created.</p> <p>If none is specified for host name and domain, the group is accessible by other containers in the same space, but is not publicly accessible.</p>
ROUTE_HOSTNAME	<p>A string or none Example: <code>myapp</code></p> <p>The host name, such as <i>mycontainerhost</i>. Do not include underscores (<code>_</code>) in the host name. The host and the domain combined form the full public route URL, such as <code>http://mycontainerhost.AppDomainName</code>.</p>

Property	Description
	<p>If a host name is not specified, a host name is created for you. If none is specified, no host name is created.</p> <p>If none is specified for host name and domain, the group is accessible by other containers in the same space, but is not publicly accessible.</p>

Click **SAVE**.

Tutorial: Deploying a Cloud Foundry Node.js app in a container

You can take an app that you deployed previously by using Cloud Foundry and deploy it in a container by using IBM Cloud Container Service.

Objectives

- Learn the general process of deploying containers

- Learn what must be included in a Dockerfile to build an image

Time required

30 minutes

Audience

Cloud Foundry app developers

Prerequisites

- [Install the IBM Cloud Container Service CLI](#)

- [Log in to the IBM Cloud Container Service CLI](#)

Lesson 1: Download Cloud Foundry Node.js app code

Get your Node.js code ready to go. Don't have any code yet? You can download starter code from Cloud to use in this tutorial.

Before you begin, [install the IBM Cloud Container Service CLI](#) and [log in](#).

To get and organize the app code:

Create a directory that is named *cf-pi-myinitials* and navigate into it. In this directory, you save all the files that are required to build the Docker image and to run your Node.js app. You can use any name for the directory. If you use the name from the example, replace *myinitials* with your initials, such as *cf-pi-ms*.

```
mkdir cf-pi-myinitials && cd cf-pi-myinitials
```

Copy your Node.js app code and all related files into the directory. You can use your own Node.js app code or download the Personal Insights app from Cloud.

To download the Personal Insights app code from Cloud, follow these steps.

In the Cloud Catalog, in **Boilerplates**, click **Personality Insights Node.js Web Starter**. This boilerplate includes a Node.js app that uses the IBM Watson™ Personality Insights service to extract a spectrum of cognitive and social characteristics from a text that a person generates through blogs, tweets, or forum posts.

Enter the app name *cf-pi-myinitials* and click **CREATE**. To access the app code for the boilerplate, you must deploy the CF app to Cloud first. You can use any name for the app. If you use the name from the example, replace *myinitials* with your initials, such as *cf-pi-ms*, to ensure that it is unique.

This deployment creates an app that is named *cf-pi-myinitials* as well as an instance of the IBM Watson Personality Insights service that is named *cf-pi-myinitials-personality_insights*.

As the app is deployed, instructions for **Deploying your app with the command line interface** are displayed.

From step 1, click **DOWNLOAD STARTER CODE**.

Extract the .zip file and save the app files to your directory.

Your Cloud Foundry app code is ready to be containerized!

Lesson 2: Creating a Docker image with your CF app code

Create a Dockerfile that includes your Node.js app code and the necessary configurations for your container. Then, build a Docker image from that Dockerfile and push it to your private Cloud registry.

To create a Docker image with your app code, follow these steps.

Create a `Dockerfile`, which is the basis for creating a container. You can create the `Dockerfile` by using your preferred CLI editor or a text editor on your computer. The following example shows how to create a `Dockerfile` file with the [nano editor](#).

```
nano Dockerfile
```

Copy the following script into the `Dockerfile`.

```
#Use the IBM Node image as a base image
FROM registry.DomainName/ibmnode:latest

#Expose the port for your Personal Insights app, and set
#it as an environment variable as expected by cf apps
ENV PORT=3000
EXPOSE 3000
ENV NODE_ENV production

#Copy all app files from the current directory into the app
#directory in your container. Set the app directory
#as the working directory
ADD . /app
WORKDIR /app

#Install any necessary requirements from package.json
RUN npm install

#Sleep before the app starts. This command ensures that the
#IBM Containers networking is finished before the app starts.
CMD (sleep 60; npm start)

#Start the Personal Insight app.
CMD ["node", "/app/app.js"]
```

Save your changes by pressing **ctrl + o**. Confirm your changes by pressing **ENTER**. Exit the nano editor by pressing **ctrl + x**.

Build the Docker image with your app code and push it to your private Cloud registry.

```
bx ic build -t registry.DomainName/namespace/cf-pi-myinitials .
```



Understanding this command's components

build	The build command.
registry.DomainName/namespace/cf-pi-myinitials	Your private Cloud registry path, which includes your unique namespace and the name of the image. For this example, the same name is used for the image as the CF node app <i>cf-pi-myinitials</i> , but you can choose any name for the image in your private registry. If you are unsure what your namespace is, run the following command to find it. <pre>bx ic namespace-get</pre>
.	The location of the Dockerfile. If you are running the build command from the directory that includes the Dockerfile, enter a period (.). Otherwise, use a relative path to the Dockerfile.

The image is created in your registry. You can run the `bx ic images` command to verify that the image was created.

```
$ bx ic images
```

REPOSITORY	TAG	IMAGE ID	CREATED
registry.DomainName/namespace/cf-pi-myinitials	latest	b02b7a24c35f	48 seconds

Lesson 3: Deploying a container from your image

Deploy your app as a container group in Cloud.

To deploy your app in a container group, follow these steps.

Create a container group from the image that you created in the previous lesson by running

the following command.

```
bx ic group-create --name cf-pi-myinitials -m 64 -n pi-app-myinitials -d
```



Understanding this command's components

group-create	The group-create command.
--name <i>cf-pi-myinitials</i>	The name of the container as you want it to display from the Cloud GUI.
-m 64	Each instance in the group is assigned 64 MB of memory.
-n <i>pi-app-myinitials</i>	The host name in the route URL. <div>Tip<p>If you are using the starter code, use a different name for the host name than what you used in the previous lesson to deploy the Cloud Foundry app. If the same host is used and the CF app is still running, load balancing occurs between the container group and app.</p></div>
-d AppDomainName	The domain name in the route URL. Unless you have a custom domain, use AppDomainName.
-p 3000	The port number to expose. Connections to the route URL are forwarded to port 3000 on the container.
-e CCS_BIND_SRV= <i>cf-pi-myinitials-personal_insights</i>	Bind the IBM Watson Personality Insights service instance to the container, so you can analyze cognitive and social characteristics from the text that you enter in the Personal



Understanding this command's components

	<p>Insights app. The service instance was created as part of the CF app deployment in lesson 1.</p> <p>If you are unsure what the name of your service instance is, run the following command.</p> <pre>bx service list</pre>
<code>--desired 3</code>	The number of single container instances to create in the container group. All of the instances are the same.
<code>--auto</code>	Enable automatic recovery for the container group. If one of the instances in the group becomes unavailable, another instance is automatically created that replaces the unresponsive one.
<code>--anti</code>	Enable anti-affinity for the container group. Each instance of the group is placed on a separate physical compute node, which reduces the odds of all containers in a group crashing due to a hardware failure.
<code>registry.DomainName/namespace/cf-pi-myinitials:latest</code>	The destination repository path, which includes your unique namespace and the name of the destination image. For this example, the same name is used for the image as the CF node app <i>cf-pi-myinitials</i> .

Wait a few minutes for the container to deploy and for the networking to be set up. When the container is deployed, you can view your Personal Insights app in a browser by entering the following URL `http://pi-app-myinitials.AppDomainName/`.

Remove the CF app. Now that your CF app code is deployed in a container, you can remove the CF app to maximize the use of your quota.

```
cf delete cf-pi-myinitials
```

Confirm the deletion by entering **y**.

Your app is deployed in a container!

CLI reference for managing single and scalable containers in IBM Cloud Container Service (Deprecated)

Refer to these commands to create and manage single and scalable containers.

Attention

Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

In this page:

- [bx ic commands](#)
- [Quick reference](#)
- [Version history](#)
- [Container states](#)
- [Supported native Docker commands](#)
- [Docker Compose commands](#)
- [Supported CLIs](#)
- [\(Deprecated\) cf ic commands](#)
- [Migrating cf ic to bx ic commands](#)

IBM Cloud Container Service plug-in (bx ic) commands for managing single and scalable containers

With the IBM Cloud Container Service plug-in, refer to these commands to create and manage containers.

The cf ic CLI has been deprecated

Instead of using the IBM Cloud Container Service plug-in for Cloud Foundry (cf ic), use the IBM Cloud Container Service plug-in for Cloud. You can continue to use the cf ic CLI until 01 January 2018.

To start using the IBM Cloud Container Service plug-in for Cloud instead, refer to the following topics.

[Installing the CLI](#)

[Installing the bx ic plug-in](#)

[bx ic plug-in reference](#)

You can run any of these commands without any options to review help information. You can also run `bx ic --help` to view a list of these commands.

Table 1. Commands for managing the CLI and your service on Cloud

Commands for managing the CLI and your service on Cloud			
<code>bx ic info</code>	<code>bx ic init</code>	<code>bx ic namespace</code>	<code>bx ic reprovision</code>
<code>bx ic unprovision</code>	<code>bx ic update</code>	<code>bx ic version</code>	

Table 2. Commands for creating and managing containers on

Commands for creating and managing containers on Cloud				
<code>bx ic attach</code>	<code>bx ic cp</code>	<code>bx ic exec</code>	<code>bx ic group-create</code>	<code>bx ic group-inspect</code>
<code>bx ic group-instances</code>	<code>bx ic group-remove</code>	<code>bx ic group-update</code>	<code>bx ic groups</code>	<code>bx ic inspect</code>
<code>bx ic ip-bind</code>	<code>bx ic ip-release</code>	<code>bx ic ip-request</code>	<code>bx ic ip-unbind</code>	<code>bx ic ips</code>
<code>bx ic kill</code>	<code>bx ic logs</code>	<code>bx ic pause</code>	<code>bx ic port</code>	<code>bx ic ps</code>
<code>bx ic rename</code>	<code>bx ic restart</code>	<code>bx ic rm</code>	<code>bx ic route-map</code>	<code>bx ic route-unmap</code>

Commands for creating and managing containers on Cloud

bx ic run	bx ic service-bind	bx ic service-unbind	bx ic start	bx ic stats
bx ic stop	bx ic top	bx ic unpause	bx ic wait	bx ic wait-status

Table 3. Commands for managing images on Cloud

Commands for managing images on Cloud				
bx ic build	bx ic cpi	bx ic images	bx ic inspect	bx ic namespace-get
bx ic namespace-set	bx ic rmi			

Table 4. Commands for storage for containers on Cloud

Commands for storage for containers on Cloud				
bx ic volume-create	bx ic volume-fs-create	bx ic volume-fs-flavors	bx ic volume-fs-inspect	bx ic volume-fs
bx ic volume-fs-remove	bx ic volume-fs-inspect	bx ic volume-remove	bx ic volumes	

bx ic attach [--no-stdin] [--sig-proxy] CONTAINER

Purpose

Control a running container or view its output. Use CTRL+C to exit and stop the container. For more information, see the [attach](#) command in the Docker help.

Parameters

--no-stdin

(Optional) Do not include the standard input.

--sig-proxy

(Optional) The default is true.

CONTAINER

(Required) A container name or ID.

Example

The following example is a request to attach to the container *my_container*.

```
bx ic attach my_container
```

bx ic build -t TAG [--no-cache] [--pull] [-q] DOCKERFILE_LOCATION

Purpose

Build a Docker image locally or in the private Cloud images registry. For more information, see the [build](#) command in the Docker help.

Parameters

DOCKERFILE_LOCATION

(Required) The path to the Dockerfile and context on the local host.

--no-cache

(Optional) When an image is built for the first time, the layers for the image are cached for an entire organization. When you build subsequent versions of the image, these cached versions of the image are used to speed up the build process. By including the `--no-cache` option when you build a new version of the image, you can ensure that each layer of the image is re-built too.

--pull

(Optional) By default, the base image is only pulled from the registry if it isn't already in your organization's image cache. By including this option, you can ensure that latest version of the base image is used to build the new image.

-q, --quiet

(Optional) Suppress the verbose output that is generated by the containers. The default is `false`.

-t TAG, --tag TAG

(Required) The path to your private images registry to apply to the image that is created.

Example

The following example is a request to build an image that is named `myimage`. The Dockerfile and other artifacts to use in the build are in the same directory that the

command is run from. Because the registry and namespace are included with the image name, the image is built in your organization's private Cloud images registry.

Tip

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

```
bx ic build -t registry.DomainName/<my_namespace>/myimage .
```

bx ic cp SRC_PATH CONTAINER:DEST_PATH | bx ic cp CONTAINER:SRC_PATH DEST_PATH

Purpose

Copy a file or folder from a running or stopped single container to your local machine and vice versa. For more information, see the [cp](#) command in the Docker help.

Note

This command is supported to be used with single containers only. When you want to copy files from or to a container group, you must copy the files from or to each container instance of the group.

Parameters

DEST_PATH

(Required) The path to the file or folder on your local machine or container where you want to copy the information from the source path. The file or folder must exist on your local machine or the container before you can start copying. If you want to copy to your local machine, you can use the absolute or relative path. If you want to copy to your single container, you must include the name of the container, the delimiter ":", and the path to the destination file or folder. The path must be relative to the root directory

of the container.

SRC_PATH

(Required) The path to the file or folder that you want to copy. If the file or folder is on your local machine, you can use the absolute or relative path. If the file or folder is in your single container, you must include the name of the container, the delimiter ":", and the path to the file or folder that you want to copy. The path must be relative to the root directory of the container.

Note

You cannot copy certain system files that are stored under the `/dev`, `/etc`, `/proc` and `/sys` folder of your container.

Example

This example command copies a file that is named `config.txt` from the current directory of your local machine to the `config` directory of a single container that is named *my_container*.

```
bx ic cp ./config.txt my_container:/config
```

bx ic cpi SRC DST

Purpose

Access a Docker Hub image or an image from your local registry and copy it to your private Cloud images registry.

Note

This command is an alternative to using `docker pull` and `docker push` together to add an image to your Cloud images registry. If you prefer to use `docker pull` and `docker push`, see [Pushing local images to your private](#)

Cloud registry from the command line.

Parameters

DST

(Required) The private Cloud images registry URL, which includes the namespace, and the destination image name. A tag for the image is optional.

SRC

(Required) The source repository and image name.

Example

```
bx ic cpi source_repository/source_image_name private_registry_URL/d
```

Tip

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

```
bx ic cpi training/sinatra registry.DomainName/<my_namespace>/mysina
```

bx ic exec [-d] [-it] [-u USER] CONTAINER CMD

Purpose

Execute a command within a container. For more information, see the [exec](#) command in the Docker help.

Parameters

CMD

(Required) The command to execute within the container or containers specified.

CONTAINER

(Required) A container name or ID.

-d , --detach

(Optional) Run the specified command in the background.

-it

(Optional) Interactive mode. Keep the standard input display. Type `exit` to exit.

-u *USER*, --user *USER*

(Optional) A user name.

Examples

This example command runs an interactive bash terminal in the container *my_container*.

```
bx ic exec -it my_container bash
```

This example command runs the `date` command in the container *my_container*.

```
bx ic exec my_container date
```

bx ic group-create [--anti] [--auto] [-d DOMAIN] [--desired DESIRED] [-e ENV] [--env-file ENVFILE] [--http_monitor_enabled] [--http_monitor_path] [--http_monitor_rc_list] [--ip IP_ADDRESS] [-m MEMORY] [--max MAX] [--min MIN] [-n HOST] --name NAME [-p PORT] [--session-affinity] [--volume VOLUME:/DIRECTORY_PATH] IMAGE [CMD]

Purpose

Create a scaling group. By default, containers in Cloud run in detached mode.

Parameters

--anti

(Optional) Use anti-affinity to make your container group more highly available. The `--anti` option forces every container instance in your group to be placed on a separate physical compute node, which reduces the odds of all containers in a group crashing due to a hardware failure. You might not be able to use this option with larger group sizes because each Cloud region and organization has a limited set of compute nodes available for deployment. If your deployment does not succeed, either reduce the

number of container instances in the group or remove the `--anti` option.

--auto

(Optional) When automatic recovery is enabled for a container group and the group has been running for 10 minutes, the group's load balancer starts to regularly check the health of each container instance in the group via HTTP requests. If a container instance does not respond within 100 seconds, it is marked as inactive. Inactive container instances are removed from the group and re-created by auto-recovery. Auto-recovery attempts to recover container instances in a group 3 times. After the third attempt, auto-recovery does not recover any container instances in the group for 60 minutes. After 60 minutes, the auto-recovery process starts again. For more information about the health check, see the `--http_monitor_enabled` option.

CMD

(Optional) The command and arguments are passed to the container group to execute. This command must be a long-running command. Do not use a short-lived command, for example, `/bin/date`, because it might cause the container to crash.

Example long-running commands using the `ibmnode` image that is provided by IBM Cloud Container Service:

```
bx ic group-create --name my_container_group registry.Domain
```

```
bx ic group-create --name my_container_group registry.Domain
```

Note

The `CMD` and its arguments must come at the end of the `bx ic group-create` command line.

-d DOMAIN, --domain DOMAIN

(Optional) The default system domain is `AppDomainName` and already provides a SSL certificate, so you can access your container groups with HTTPS without any additional configuration. To use a custom domain, you

must register the custom domain on a public DNS server, configure the custom domain in Cloud, and then map the custom domain to the Cloud system domain on the public DNS server. After your custom domain is mapped to the Cloud system domain, requests for your custom domain are routed to your application in Cloud. When you create a custom domain, do not include underscores (`_`) in the domain name. For more information about custom domains, see [Creating and using a custom domain](#). To make your custom domain secure, [upload a SSL certificate](#), so your container groups can be accessed with HTTPS.

The host and the domain combined form the full public route URL, such as `http://mycontainerhost.AppDomainName` and must be unique within Cloud. When you review the details of a container group with the `bx ic group -inspect` command, the host and the domain are listed together as the route.

--desired *DESIRED*

(Optional) The number of instances in your group. The default is 2. The number of container group instances that your app requires depends on the expected workload and the time in which a request can be processed by the app. For example, multi-tasking or network connectivity affects how many instances might be required for your app. To determine the number of desired instances, deploy a single container in Cloud that runs your app and perform a load test on this container. If the app can handle, for example, 100 requests per second, but you are expecting an average workload of 300 requests per second, then the desired number of instances for the container group is 3. To make your container group highly available and more resilient to failure, consider including extra instances than the minimum to handle the expected workload. Extra instances can handle the workload in case another instance crashes. To ensure automatic recovery of a failed instance without affecting the workload, include one extra instance. For protection against two simultaneous failures, include two extra instances. This set up is an $N+2$ pattern, where N is the number of instances to handle the requests and $+2$ is an extra two instances.

Important

A container group should include at least 2 container instances. If you include only 1 instance in the group and that instance becomes unavailable, your users can experience downtime for the app.

-e ENV, --env ENV

(Optional) Set the environment variable where ENV is a key=value pair. List multiple keys separately and if you include quotation marks, include them around both the environment variable name and the value. Example:

```
-e "key1=value1" -e "key2=value2a,value2b" -e "key3=value3"
```

Table 5. Suggested environment variables

Environment variable	Description
CCS_BIND_APP= <appname>	Some Cloud services do not support direct binding to a container. In this case, you need to create a Cloud Foundry app and bind the Cloud service to it. Then, you bind the app to your container by using CCS_BIND_APP. The Cloud Foundry app acts as a bridge and allows Cloud to inject your bridge app's VCAP_SERVICES information into the running container instance.
CCS_BIND_SRV= <service_instance_name1>, <service_instance_name2>	To bind a Cloud service directly to a container without using a bridge app, use CCS_BIND_SRV. This binding allows Cloud to inject the VCAP_SERVICES information into the running container instance. To list multiple Cloud services, include them as part of the same environment variable.

Note
When a service does not

Environment variable	Description
support the use of the CCS_BIND_SRV= environment variable, use CCS_BIND_APP= instead.	
(Deprecated) CCS_SSH_KEY= <public_ssh_key>	<p>This environment variable has been deprecated</p> <p>Use <code>bx ic exec</code> or <code>bx ic attach</code> for external access to your containers instead. For more information, see Logging in to a container with exec.</p> <p>To add an SSH key to a container when you create it, you can use CCS_API_KEY.</p>
LOG_LOCATIONS= <path_to_file>	To add a log file to be monitored in the container, include the LOG_LOCATIONS environment variable with a path to the log file.

Example:

```
bx ic group-create -e CCS_BIND_SRV=<service_instance_name> -
```

--env-file *ENVFILE*

(Optional) Import environment variables from a file where *ENVFILE* is the path to your file on your local directory. Every line in the file represents one key=value pair.

Example

```
KEY1=VALUE1  
KEY2=VALUE2
```

--http_monitor_enabled

(Optional) The default value is `true`. When automatic recovery is enabled for a container group and the group has been running for 10 minutes, the group's load balancer starts to regularly check the health of each container instance in the group via HTTP requests. If a container instance does not respond within 100 seconds, it is marked as inactive. Inactive container instances are removed from the group and re-created by auto-recovery. Auto-recovery attempts to recover container instances in a group 3 times. After the third attempt, auto-recovery does not recover any container instances in the group for 60 minutes. After 60 minutes, the auto-recovery process starts again. Some apps might not require an HTTP health check, such as an app that uses TCP for communication instead of HTTP. To disable the HTTP monitor, set the option to `false`. Example:

```
--http_monitor_enabled=false
```

--http_monitor_path

(Optional) The default value is `/`. When `--http_monitor_enabled` is enabled, you can use this option to set a custom path for the health check, such as `--http_monitor_path="/health/path"`.

--http_monitor_rc_list

(Optional) When `--http_monitor_enabled` is enabled, you can specify the list of response codes for the health check. The default value includes the 200, 201, 202, 204, 300, 301, 302, 401, 403, and 404 codes. Any response not in the list of valid responses changes the container instance's state to `INACTIVE` by the load balancer. An inactive container does not receive any load and the state of the container group changes to `HAS_INACTIVE_INSTANCES`. If none of the containers are `ACTIVE`, then the state of the container group changes to `NO_ACTIVE_INSTANCES`. To learn more about states, see [Container states in IBM Cloud Container Service](#).

IMAGE

(Required) The image to include in each container instance in the container

group. You can list commands after the image, but do not put any options after the image; include all options before you specify an image.

If you are using an image in your organization's private Cloud images registry, specify the image in the following format:

`registry.DomainName/NAMESPACE/IMAGE`

If you are using an image that is provided by IBM Cloud Container Service, do not include your organization's namespace. Specify the image in the following format: `registry.DomainName/IMAGE`

--ip *IP_ADDRESS*

(Optional) If you have an available IP address, bind a public IP address to your container group. You must also expose the port with the `-p` flag. A container group can have either a public route or a public IP address but not both. To unbind the IP address and return the IP to the space's quota, remove the container group and recreate the group without the IP address. To request an IP address prior to creating a container group, run `bx ic ip-request`.

Note

Binding a public IP address does not include a SSL certificate which encrypts the data that are sent to and from your container group. If your app requires SSL encryption, you can either implement your own SSL solution or map a public route to your container group instead of binding a public IP address. Public routes already include a SSL certificate, so you can access your container group with HTTPS without any additional configuration.

-m *MEMORY*, --memory *MEMORY*

(Optional) Enter a memory limit for your container in MB. The memory limit is part of the container size that defines the maximum amount of memory and disk space a container gets on the compute host during runtime. After a container size is assigned, the value cannot be changed. Available sizes in IBM Cloud Container Service are:

Pico (64 MB memory, 4 GB disk space)

Nano (128 MB memory, 8 GB disk space)

Micro (256 MB memory, 16 GB disk space)

Tiny (512 MB memory, 32 GB disk space)

Small (1024 MB memory, 64 GB disk space)

Medium (2048 MB memory, 128 GB disk space)

Large (4096 MB memory, 256 GB disk space)

X-Large (8192 MB memory, 512 GB disk space)

2X-Large (16384 MB memory, 1 TB disk space)

If you do not set a size for your container, each container instance is created with 256 MB.

Important

Enter the container memory in MB without the unit label. For example, if you want to create a pico container, enter `-m 64`.

--max *MAX*

(Optional) The maximum number of instances. The default is 2.

--min *MIN*

(Optional) The minimum number of instances. The default is 1.

-n *HOST*, --hostname *HOST*

(Optional) The host name, such as *mycontainerhost*. Do not include underscores (`_`) in the host name. The host and the domain combined form the full public route URL, such as `http://mycontainerhost.AppDomainName`. When you review the details of a container group with the `bx ic group-inspect` command, the host and the domain are listed together as the route.

--name *NAME*

(Required) Assign a name to the group. `-n` is deprecated.

Tip

The container name must start with a letter, and then can include uppercase letters, lowercase letters, numbers, periods (.), underscores (_), or hyphens (-).

-p *PORT*, --publish *PORT*

(Required) Expose the port for HTTP and private network traffic. If you do not expose a port, the container group runs, but the container group cannot be accessed through the group load balancer on either the public or private network.

For container groups, you cannot include multiple ports. If multiple ports are listed, only the last listed port is exposed. The other ports are ignored by the command.

When you bind a route, containers in your group must listen for HTTP traffic on the group's exposed port. Non-HTTP ports cannot be exposed publicly. When HTTPS traffic arrives on the exposed port, the (Go)Router completes the HTTPS termination. Then, the HTTP protocol is used on the private virtual network between the (Go)Router and the containers. When you bind an IP address, containers in your group can listen for HTTP or HTTPs traffic, but you can still expose only one port.

If a port is specified in the Dockerfile for the image that you are using, include that port.

Tip

For the IBM certified Liberty Server image or a modified version of this image, enter the port 9080.

For the IBM certified Node.js image or a modified version of this image, enter any port, such as port 8000.

--session_affinity

(Optional) The default value is `false`. When `--http_monitor_enabled` is set to `true`, you can set the `--session_affinity` option to `true` to allow IBM Cloud Container Service to create and manage a session cookie on behalf of the app. This option is not available if `--http_monitor_enabled` is set to `false` or if you are using TCP for communication instead of HTTP.

--volume VOLUME:/DIRECTORY_PATH[:ro]

(Optional) Attach a volume to a container by specifying the details in the following format `VOLUME:/DIRECTORY_PATH[:ro]`.

VOLUME: The volume ID or name.

DIRECTORY_PATH: The absolute path to the directory in the container.

`ro`: Optional. Specifying `ro` makes the volume read-only instead of the default read/write.

Example

The following example is a request to create a scaling group with a public route.

```
bx ic group-create --auto -d AppDomainName -n mycontainerhost --nam
```

The following example is a request to create a scaling group with a public IP address.

```
bx ic group-create --auto --ip 198.51.100.27 --name my_container_gr
```

bx ic group-inspect GROUP

Purpose

See the detailed information that was specified for a container group when it was created, such as the environment variables, ports, or memory. Also, run this command to see the public route that is assigned to the group.

Parameters

GROUP

(Required) The group ID or name.

Example

The following example is a request to inspect the group *my_container_group*.

```
bx ic group-inspect my_container_group
```

bx ic group-instances GROUP

Purpose

List all containers with their ID, name, IP address and public port that are part of a container group.

Parameters

GROUP

(Required) The group ID or name.

Example

The following example is a request to retrieve all containers of the group *my_container_group*.

```
bx ic group-instances my_container_group
```

bx ic group-remove [-f] GROUP [GROUP]

Purpose

Remove one or more container groups from a space.

Parameters

-f, --force

(Optional) Forces the removal of a running or failed container.

GROUP

(Required) The group ID or name of at least one container group.

Example

The following example is a request to remove a container group, where *my_container_group* is the name of the group.

```
bx ic group-remove my_container_group
```

bx ic group-update [--desired DESIRED] [-e "KEY=VALUE"] [--max MAX] [--min MIN] GROUP

Purpose

Use the `bx ic group-update` command to perform the following updates on your group.

Update the desired number of instances that run in your container group.

Add new environment variables to the group.

Update the value of an existing environment variable.

Note

You can perform one update at a time only.

Tip

To update the host name or domain for a container group, use `bx ic route-map [-d DOMAIN] [-n HOST] GROUP`.

Parameters

--desired *DESIRED*

(Optional) The number of instances that you require. The default is 2.

-e "*KEY=VALUE*"

(Optional) Add a new environment variable to the container group or change the value of an existing environment variable.

--max *MAX*

(Optional) The maximum number of instances. The default is 2.

--min *MIN*

(Optional) The minimum number of instances. The default is 1.

Example

The following example is a request to update the desired number of container instances in a container group named *my_container_group*.

```
bx ic group-update --desired 5 my_container_group
```

The following example is a request to add a new environment variable *environment* with the value *Linux* to a container group that is named *my_container_group*.

```
bx ic group-update -e "environment=Linux" my_container_group
```

bx ic groups [-q]

Purpose

List the container groups in a space.

Parameters

-q

(Optional) List the container group ID of each container group only.

Example

```
bx ic groups [-q]
```

bx ic images [-a] [-f **CONDITION**] [--no-trunc] [q]

Purpose

See a list of all the available images in the organization's private Cloud images registry. For more information, see the [images](#) command in the Docker help.

Parameters

-a, --all

(Optional) Includes all of the image layers for each image in your organization's registry, not just the most recent layer.

-f, --filter *CONDITION*

(Optional) Filter the list of images by the condition provided.

--no-trunc

(Optional) Do not truncate the output.

-q, --quiet

(Optional) Display the numeric IDs only.

Responses

You receive a list of images that includes the image ID, the created date, and the image name.

Example

The following example is a request to receive a list of available images for the organization.

```
bx ic images
```

bx ic info [--json]

Purpose

Receive a set of information that describes the state of the container cloud service instance.

Parameters

--json

(Optional) Structure the output of the command in JSON format.

Responses

You receive a readout that includes information about the containers limit, containers usage, containers that are running, memory limit, memory usage, floating IP limit, floating IP usage, CCS host URL, registry host URL, and debug mode status.

Example

The following example is a request to receive information about the container cloud service instance.

```
bx ic info
```

bx ic init

Purpose

Initialize the IBM Cloud Container Service CLI.

Note

Before logging in, you must ensure that the Cloud Foundry command tool (cf) is installed, as well as the Docker CLI (docker), and that they are both configured in your environment path.

Tip

To switch to another region, specify the API endpoint for that region when you log in to the Cloud Foundry CLI, not the IBM Cloud Container Service CLI.

US South:

```
bx login -a api.ng.bluemix.net
```

United Kingdom:

```
bx login -a api.eu-gb.bluemix.net
```

Example

```
bx ic init
```

bx ic inspect [IMAGE | images | CONTAINER]

Purpose

Retrieve information about one or more containers or container images. Only one of IMAGE, images, or CONTAINER options can be specified at a time. For more information, see the [inspect](#) command in the Docker help.

Parameters

CONTAINER

(Required) Get detailed information about a specific container by specifying the container name or ID. You can retrieve information for multiple containers by listing their names or IDs in the command with a space in between.

IMAGE

(Required) Get detailed information about a specific image by specifying the full private Cloud registry path to your image. You can retrieve information for multiple images by listing each registry path in the command with a space in between

images

(Required) Get detailed information about all of the images in your registry.

Example

The following example is a request to inspect a container that is named `my_container`.

```
bx ic inspect my_container
```

bx ic ip-bind ADDRESS CONTAINER

Purpose

Bind an available floating IP address to a single container.

Note

To bind a floating IP address to a container group, use the `--ip` flag in the [bx](#)

`ic group-create` command.

Parameters

ADDRESS

(Required) The IP address to bind to the single container.

CONTAINER

(Required) The container ID or name to bind to the IP address.

Example

The following example is a request to bind the IP address 192.0.2.23 to the container with name *my_container*.

```
bx ic ip-bind 192.0.2.23 my_container
```

bx ic ip-release ADDRESS

Purpose

Release a floating IP address. After a floating IP address is released, it cannot be bound to a single container or container group in the space.

Parameters

ADDRESS

(Required) The IP address to release.

Example

The following example is a request to release the IP address 192.0.2.23

```
bx ic ip-release 192.0.2.23
```

bx ic ip-request [-q]

Purpose

Request a new floating IP.

Parameters

-q

(Optional) Only the IP address is displayed in the command output. No

message text is included.

Example

The following example is a request for a new floating IP address.

```
bx ic ip-request
```

bx ic ip-unbind ADDRESS CONTAINER

Purpose

Unbind a floating IP address from a single container.

Note

To unbind the IP address from a container group, remove the container group and recreate the group without the IP address.

Parameters

ADDRESS

(Required) The IP address that is being unbound.

CONTAINER

(Required) The container ID or name that is being unbound.

Example

The following example is a request to unbind the IP address 192.0.2.23 from the container with name *my_container*.

```
bx ic ip-unbind 192.0.2.23 my_container
```

bx ic ips [-q]

Purpose

List the available floating IP addresses for the organization and the IDs for the single containers or groups that are bound to those IP addresses.

Parameters

-q

(Optional) List only the IP addresses, without the IDs for the single containers that are bound to those IP addresses.

Responses

You receive a list of IP addresses and the ID of the single container they are linked to. If the IP address is unused, no container ID is shown.

Example

The following example is a request to receive a list of all IP addresses for the organization, whether they are available to use or not.

```
bx ic ip-list -q
```

bx ic kill [-s CMD] CONTAINER [CONTAINER]

Purpose

Stop a running process in one or more containers without stopping the container. For more information, see the [kill](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container ID or name of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

-s, --signal CMD

(Optional) Send any command to the process that is running in the container.

Example

The following example is a request to kill the process in a container that is named *my_container*.

```
bx ic kill my_container
```

bx ic logs [-f | --since TIMESTAMP | --tail STRING | -t] CONTAINER

Purpose

Show the output or error logs for a running container. For more information, see the [logs](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID.

-f, --follow

(Optional) Include new output from the STDOUT and STDERR log files for the container.

--since *TIMESTAMP*

(Optional) Show all logs since the timestamp.

--tail *STRING*

(Optional) Specify the number of the most recent lines of logs to show. The default is to show all logs.

-t, --timestamps

(Optional) Display timestamps on the logs.

Example

The following example is a request to get the output log for a container that is named *my_container*.

```
bx ic logs -o my_container
```

bx ic namespace-[get | set] NAME

Purpose

Set or retrieve the name of the private Cloud image registry for the organization that the user has logged in with.

Parameters

get

(Optional) Request the namespace for your organization's private images registry.

set *NAME*

(Optional) A one-time only function to set the namespace for your organization's private image registry, if it is not already set. After you set the

namespace, log out and in to the IBM Cloud Container Service CLI before you continue.

Restriction

You cannot use a hyphen (-) in the name of your registry namespace.

Example

The following example is a request to query the namespace for the organization that the user logged in to.

```
bx ic namespace-get
```

bx ic pause CONTAINER [CONTAINER]

Purpose

Pause all processes within one or more running containers. For more information, see the [pause](#) command in the Docker help. To start the container again, run the [unpause](#) command.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

Example

The following example is a request to pause a container that is named *my_container*.

```
bx ic pause my_container
```

bx ic port CONTAINER [PRIVATE_PORT[/PROTOCOL]]

Purpose

List all port mappings for a container or retrieve the public port and IP address that is mapped to your private port. For more information, see the [port](#) command in the

Docker help.

Parameters

CONTAINER

(Required) The container ID or name.

PRIVATE PORT

(Optional) The private port for which you want to look up the assigned public port and IP address.

PROTOCOL

(Optional) The protocol that was assigned to your private port. It can be either *tcp* or *udp*.

Example

The following example is a request to list all port and IP mappings for a container that is named *my_container*.

```
bx ic port my_container
```

System Output

```
9080/tcp -> 192.0.2.25:9080
```

In this example, 9080/tcp is your private port that is using the TCP/IP protocol. It is mapped to the public IP address 192.0.2.25 and public port 9080.

bx ic ps [-a] [--filter env=SEARCH_CRITERIA] [-l NUM] [-q] [-s]

Purpose

Retrieve a list of containers that are running in the space. By default, only running containers are displayed. For more information, see the [ps](#) command in the Docker help.

Parameters

-a, --all

(Optional) Show all containers in the space, both running and stopped.

--filter env=SEARCH_CRITERIA

(Optional) Search containers that have a specific environment variable value. You can filter your containers by any environment variable key or

value that is listed in the *Env* section of your CLI response when you inspect a container. Replace *SEARCH_CRITERIA* with the key or value you are looking for. Your search criteria does not need to be an exact match.

-l NUM, --limit NUM

(Optional) List the most recently created containers, where *NUM* is the number of the most recently created containers that you want to return.

For example, if you created containers node1 through to node5 sequentially, the option `--limit 2` returns node4 and node5 because they are the last two containers that were created.

-q, --quiet

(Optional) Display only container IDs.

-s, --size

(Optional) List the sizes of the containers.

Examples

The following example is a request to see all running and stopped containers.

```
bx ic ps -a
```

bx ic rename OLD_NAME NEW_NAME

Purpose

Rename a container. For more information, see the [rename](#) command in the Docker help.

Parameters

NEW_NAME

(Required) A new name for the container.

OLD_NAME

(Required) The old name of the container.

Example

The following example is a request to rename a container that is currently named *my_container* to *container_renamed*.

```
bx ic rename my_container container_renamed
```

bx ic reprovision [-f] [AVAILABILITY_ZONE]

Purpose

Change the IBM Cloud Container Service availability zone for a space. The original quota for the space is maintained. By deploying your app in multiple container groups in different availability zones, you are creating higher availability for your app in case of hardware failures.

Attention

When you run this command, all of your single containers and groups, public IP addresses and volumes in this space will not be migrated to the new availability zone and will be removed during the migration process. Images will not be affected. [Backup your volumes](#) to ensure you can restore data after you migrated to the new availability zone.

Parameters

AVAILABILITY_ZONE

(Optional) The name of the IBM Cloud Container Service availability zone where your containers are deployed. If no IBM Cloud Container Service availability zone is specified, the default IBM Cloud Container Service availability zone that is set for the region is used.

Important

If an IBM Cloud Container Service availability zone is included, the `-f` option must also be included.

Table 6. IBM Cloud Container Service availability zones

Region	Data center location	IBM Cloud Container Service availability zone names
--------	----------------------	---

Region	Data center location	IBM Cloud Container Service availability zone names
US South	Dallas 1	dal09-01
US South	Dallas 2	dal09-02
US South	Dallas 3	dal10-03
United Kingdom	London	lon02-01
United Kingdom	Amsterdam	ams03-01

-f, --force

(Optional) Forces the re-creation of IBM Cloud Container Service in the Cloud space.

Example

The following example is a request to change the IBM Cloud Container Service availability zone for the current space to dal09-02.

```
bx ic reprovision -f dal09-02
```

bx ic restart CONTAINER [CONTAINER] [-t SECS]

Purpose

Restart one or more containers. For more information, see the [restart](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

-t SECS, --time SECS

The number of seconds to wait before the container is restarted.

Example

The following example is a request to restart a container that is named *my_container*.

```
bx ic restart my_container
```

bx ic rm [-f] CONTAINER [CONTAINER]

Purpose

Remove one or more containers. For more information, see the [rm](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

-f, --force

(Optional) Forces the removal of a running or failed container.

Example

The following example is a request to remove a container that is named *my_container*.

```
bx ic rm my_container
```

bx ic rmi [-R REGISTRY] IMAGE [IMAGE]

Purpose

Remove one or more images from the logged-in organization's private Cloud images registry. For more information, see the [rmi](#) command in the Docker help.

Parameters

IMAGE

(Required) The full private Cloud registry path to the image that you want to remove. If a tag is not specified in the image path, the image tagged `latest` is deleted by default. You can run this command with multiple images by listing each private Cloud registry path in the command with a space in between.

-R REGISTRY, --registry REGISTRY

(Optional) Change the registry host. The default is to use the registry that you used with the **init** command.

Example

The following example is a request to remove the image `mynamespace/myimage:latest`:

```
bx ic rmi registry.DomainName/mynamespace/myimage:latest
```

bx ic route-map [-d DOMAIN] [-n HOST] GROUP

Purpose

Establish the route for internet traffic to use to access a container group. You can use this command to establish a new route or update an existing route.

Parameters

-d DOMAIN, --domain DOMAIN

(Optional) The domain name for the route, which is the second part of the full public route URL. In most cases, the domain is `AppDomainName`. You can also use this parameter to specify a custom domain. If you use a custom domain, do not include underscores (`_`) in the domain name.

GROUP

(Required) The group ID or name.

-n HOST, --hostname HOST

(Optional) The host name for the route. Do not include underscores (`_`) in the host name. The host name is the first part of the full public route URL, such as `mycontainerhost` in the URL `mycontainerhost.AppDomainName`.

Example

The following example is a request to map the route for a container group that is named `my_container_group`. The host name is `my_host` and `AppDomainName` is the domain.

```
bx ic route-map -d AppDomainName -n my_host my_container_group
```

bx ic route-unmap [-d DOMAIN] [-n HOST] GROUP

Purpose

Unmap the route from a container group. A group that does not have a route is not accessible from the Internet.

Parameters

-d DOMAIN, --domain DOMAIN

(Optional) The domain name for the route.

GROUP

(Required) The group ID or name.

-n HOST, --hostname HOST

(Optional) The host name for the route.

Example

The following example is a request to unmap the route for the group that is called `my_container_group`, where `my_host` is the host name and `AppDomainName` is the domain.

```
bx ic route-unmap -d AppDomainName -n my_host my_container_group
```

bx ic run [-p PORT] [-P] [-d] [-e ENV] [--env-file ENVFILE] [-it] [--link NAME:ALIAS] [-m MEMORY] --name NAME [--volume VOLUME:/DIRECTORY_PATH] IMAGE [CMD [CMD ...]]

Purpose

Start a new container in the container cloud service from an image name. By default, container in Cloud run in detached mode. For more information, see the [run](#) command in the Docker help.

Note

You must ensure that the Cloud Foundry command tool is installed and that you have a Cloud Foundry token. Successful login by using `bx login` and `bx ic init` generates the required token.

Parameters

CMD

(Optional) The command and arguments are passed the container to execute. This command must be a long-running command. Do not use a short-lived command, for example, **/bin/date**, because it might cause the container to crash.

Example long-running commands, assuming that you are using the `ibmnode` image that is provided in the IBM Cloud Container Service registry:

```
bx ic run --name my_container registry.DomainName/ibmnode pi
```

```
bx ic run --name my_container registry.DomainName/ibmnode --
```

```
bx ic run --name my_container registry.DomainName/ibmnode --
```

Note

The CMD and its arguments must come at the end of the `bx ic group-create` command line.

-d

(Optional) Run the container in detached mode. Containers run in detached mode by default when you use `bx ic run`. In detached mode, when the root process in the container exits, the container also exits. For example, when you deploy a container from the `ibmnode` image, your container shuts down immediately after it is built as no long running process exists.

Tip

If you choose to use `docker run` instead of `bx ic run`, the `-d` option is required.

-e ENV, --env ENV

(Optional) Set a number of environment variables where ENV is a key=value pair. List multiple keys separately and if you include quotation

marks, include them around both the environment variable name and the value.

Example

```
-e "key1=value1" -e "key2=value2" -e "key3=value3"
```

Table 7. Suggested environment variables

Environment variable	Description
CCS_BIND_APP= <appname>	Some Cloud services do not support direct binding to a container. In this case, you need to create a Cloud Foundry app and bind the Cloud service to it. Then, you bind the app to your container by using CCS_BIND_APP. The Cloud Foundry app acts as a bridge and allows Cloud to inject your bridge app's VCAP_SERVICES information into the running container instance.
CCS_BIND_SRV= <service_instance_name1>, <service_instance_name2>	To bind a Cloud service directly to a container without using a bridge app, use CCS_BIND_SRV. This binding allows Cloud to inject the VCAP_SERVICES information into the running container instance. To list multiple Cloud services, include them as part of the same environment variable.
<div>Note</div> <div>When a service does not support the use of the CCS_BIND_SRV= environment variable, use CCS_BIND_APP= instead.</div>	
(Deprecated) CCS_SSH_KEY= <public_ssh_key>	This environment variable

Environment variable	Description
	<p>has been deprecated</p> <p>Use <code>bx ic exec</code> or <code>bx ic attach</code> for external access to your containers instead. For more information, see Logging in to a container with exec.</p> <p>To add an SSH key to a container when you create it, you can use <code>CCS_API_KEY</code>.</p>
<code>LOG_LOCATIONS=<path_to_file></code>	To add a log file to be monitored in the container, include the <code>LOG_LOCATIONS</code> environment variable with a path to the log file.

--env-file *ENVFILE*

(Optional) Import environment variables from a file where *ENVFILE* is the path to your file on your local directory. Every line in the file represents one key=value pair.

Example

```
KEY1=VALUE1
KEY2=VALUE2
```

IMAGE

(Required) The image to include in the container. You can list commands after the image, but do not put any options after the image; include all options before you specify an image.

If you are using an image in your organization's private Cloud registry, specify the image in the following format:

`registry.DomainName/NAMESPACE/IMAGE`

If you are using an image that is provided by IBM Cloud Container Service, specify the image in the following format: `registry.DomainName/IMAGE`

-it

(Optional) Run the container in interactive mode. After the container is created, keep the standard input display. Type `exit` to exit.

--link NAME:ALIAS

(Optional) Whenever you want a container to communicate with another container that is running, you can address it using an alias for the host name. For more information, see [Linking single containers with the command line interface \(CLI\)](#).

-m MEMORY, --memory MEMORY

(Optional) Enter a memory limit for your container in MB. The memory limit is part of the container size that defines the maximum amount of memory and disk space a container gets on the compute host during runtime. After a container size is assigned, the value cannot be changed. Available sizes in IBM Cloud Container Service are:

Pico (64 MB memory, 4 GB disk space)

Nano (128 MB memory, 8 GB disk space)

Micro (256 MB memory, 16 GB disk space)

Tiny (512 MB memory, 32 GB disk space)

Small (1024 MB memory, 64 GB disk space)

Medium (2048 MB memory, 128 GB disk space)

Large (4096 MB memory, 256 GB disk space)

X-Large (8192 MB memory, 512 GB disk space)

2X-Large (16384 MB memory, 1 TB disk space)

If you do not set a size for your container, each container instance is created with 256 MB.

Important

Enter the container memory in MB without the unit label. For example, if you want to create a pico container, enter `-m 64`.

--name *NAME*

(Required) Assign a name to the container.

Tip

The container name must start with a letter, and then can include uppercase letters, lowercase letters, numbers, periods (.), underscores (_), or hyphens (-).

-p *PORT*, --publish *PORT*

(Optional) If you want to make your app accessible from the Internet, you must expose a public port. When you expose a public port, you create a Public Network Security Group for your container that allows you to send and receive public data on the exposed port only. All other public ports are closed and cannot be used to access your app from the internet. You can include multiple ports with multiple `-p` options. Ports cannot be mapped or forwarded. If you do not expose a port, your container is accessible from the private container network only. You can use the assigned private IP address to communicate with your container on the private network.

If a port is specified in the Dockerfile for the image that you are using, include that port.

Tip

For the IBM certified Liberty Server image or a modified version of this image, enter the port 9080.

For the IBM certified Node.js image or a modified version of this image, enter any port, such as port 8000.

You can choose between UDP and TCP to indicate the IP protocol that you want to use. If you do not specify a protocol, your port is automatically exposed for TCP traffic.

Examples

```
-p 9080  
-p 9080/udp  
-p 9080/udp -o 8000/tcp
```

If you have an existing IP address in the space that you want to bind to the container, you can specify the IP address with the port information. Include the IP address, the port, or the IP address and the container port number, such as `<ip-address>:<port_on_IP>:<container-port>`. The port on the IP address must match the container port exposed in the Dockerfile.

Example

```
-p 192.0.2.23:9080:9080
```

Note

Binding a public IP address does not include a SSL certificate which encrypts the data that are sent to and from your single container. If your app requires SSL encryption, you can either implement your own SSL solution or use a container group instead of a single container. Container groups are bound to a public route that already includes a

SSL certificate, so you can access your container group with HTTPS without any additional configuration. For more information, see [Creating a container group with the GUI](#).

-P

(Optional) Automatically expose the ports that are specified in the image's Dockerfile for HTTP traffic.

--volume VOLUME:/DIRECTORY_PATH[:ro]

(Optional) Attach a volume to a container by specifying the details in the following format `VOLUME: /DIRECTORY_PATH[:ro]`.

VOLUME The volume ID or name.

DIRECTORY_PATH: The absolute path to the directories in the container.

ro Optional. Specifying *ro* makes the volume read-only instead of the default read/write.

Examples

The following example is a valid request to create and start a new container by using the `my_namespace/nginx` image, with the name `my_container` and a 1024 MB memory limit, where `my_namespace` is the logged in users associated namespace.

```
bx ic run -m 1024 --name my_container registry.DomainName/my_namespa
```

The following example is a valid request to create and start a new container by using the `my_namespace/blog` image, passing in some credentials as environment variables, where `my_namespace` is the logged in users associated namespace.

```
bx ic run -e USER=johnsmith -e PASS=password --name my_container reg
```

The following example is a valid request to add a volume to a container by using the `my_namespace/blog` image, where `my_namespace` is the logged in users associated namespace.

```
bx ic run --name my_container --volume VolId1:/first/path --volume V
```

bx ic service-bind GROUP SERVICE_INSTANCE

Purpose

Add a service to a running container group. This command is only available to container groups. Single containers must bind a service as part of the `bx ic run` command. To unbind a service, see the [service-unbind](#) command.

Parameters

GROUP

(Required) The group ID or name.

SERVICE_INSTANCE

(Required) The name of the service instance to be added to the container group.

Example

The following example is a request to bind an instance of the Data Cache service named `MyDataCache` to the container *my_container_group*.

```
bx ic service-bind my_container_group MyDataCache
```

bx ic service-unbind GROUP SERVICE_INSTANCE

Purpose

Remove a service from a running container group. This command is only available to container groups. Single containers must remove the container and create a new container without the service. To bind a service, see the [service-bind](#) command.

Parameters

GROUP

(Required) The group ID or name.

SERVICE_INSTANCE

(Required) The name of the service instance to be removed from the container group.

Example

The following example is a request to unbind an instance of the Data Cache service named `MyDataCache` from the container *my_container_group*.

```
bx ic service-unbind my_container_group MyDataCache
```

bx ic start CONTAINER [CONTAINER]

Purpose

Start one or more stopped containers. For more information, see the [start](#) command in the Docker help. To stop a container, see the [stop](#) command.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

Example

The following example is a request to start a container that is named *my_container*.

```
bx ic start my_container
```

bx ic stats --no-stream CONTAINER [CONTAINER]

Purpose

For one or more containers, view live usage statistics for the container resources. Data is displayed as it is generated in real time. Use CTRL+C to exit. For more information, see the [stats](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID.

--no-stream

(Required) Display the latest results. No new information that is generated after running the command is displayed.

Example

The following example is a request for the most recent statistics about a container.

```
bx ic stats --no-stream my_container
```

bx ic stop CONTAINER [CONTAINER] [-t SECS]

Purpose

Stop one or more running containers. For more information, see the [stop](#) command in the Docker help. To start a container, see the [start](#) command.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names and IDs in the command with a space in between.

-t SECS, --time SECS

The number of seconds to wait before the container is killed.

Example

The following example is a request to stop a container that is named *my_container*.

```
bx ic stop my_container
```

bx ic top CONTAINER [CONTAINER] [ps options]

Purpose

List the processes that are running in the container.

Other than the native `top` command in Linux, the `bx ic top` command displays the output of the Linux `ps` command for the main processes that are running inside a container. As a consequence, the memory and CPU usage is not updated in your CLI output when it changes for a process.

Note

If you need to see the output of the native Linux `top` command, [exec](#) in to your container and run the `top` command.

For more information, see the [top](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID.

***ps* options**

(Optional) Use the options that are provided by the native Linux `ps` command to further detail or filter the output of the `bx ic top` command.

Example

The following example is a request to display detailed information for all current processes that are running inside a container named *my_container*.

```
bx ic top my_container -aux
```

bx ic unpause CONTAINER [CONTAINER]

Purpose

Unpause all processes within one or more running containers. For more information, see the [unpause](#) command in the Docker help. To pause a container, see the [pause](#) command.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

Example

The following example is a request to unpause a container that is named *my_container*.

```
bx ic unpause my_container
```

bx ic unprovision [-f]

Purpose

Delete IBM Cloud Container Service from the Cloud space that you are logged into.

Attention

When you run this command, all your single containers and container groups are lost. Your space is still available in Cloud. To start using IBM Cloud Container Service again, you must run `bx ic init` to initialize the IBM Cloud Container Service again.

Parameters

-f, --force

(Optional) Forces the deletion of IBM Cloud Container Service from the Cloud space.

Example

The following example is a request to delete IBM Cloud Container Service from the space.

```
bx ic unprovision
```

bx ic update

Purpose

Check to see if there is a more recent version of the IBM Cloud Container Service plug-in than the version you have installed.

Example

```
bx ic update
```

bx ic version

Purpose

Shows the version of Docker and the IBM Cloud Container Service API. To see the version of the installed plug-in for IBM Cloud Container Service, run `cf plugins` or `bx ic info`. For more information, see the [version](#) command in the Docker help.

Example

```
bx ic version
```

bx ic volume-create VOLNAME [FSNAME]

Purpose

Create the volume.

Parameters

FSNAME

(Optional) The file share name. If no file share is available or named the volume will be built on the space's default file share.

VOLNAME

(Required) The volume name. The name can contain uppercase letters, lowercase letters, numbers, underscores (_), and hyphens (-).

Example

The following example is a request to create a volume.

```
bx ic volume-create my_volume my_fileshare
```

bx ic volume-fs-create FSNAME FSSIZE FSIOPS

Purpose

Create a file share.

Parameters

FSIOPS

(Required) The file share IOPS. Valid values are 0.25, 2 or 4 IOPS per GB.

FSNAME

(Required) The file share name. The name can contain uppercase letters, lowercase letters, numbers, underscores (_), and hyphens (-).

FSSIZE

(Required) The file share system size in GB. Use `bx ic volume-fs-flavors` to list all valid sizes

Example

```
bx ic volume-fs-create my_fileshare 20 0.25
```

bx ic volume-fs-flavors

Purpose

Lists the file share sizes that are available to use.

Example

```
bx ic volume-fs-flavors
```

bx ic volume-fs-inspect FSNAME

Purpose

Inspect the file share.

Parameters

FSNAME

(Required) The file share name.

Example

```
bx ic volume-fs-inspect my_fileshare
```

bx ic volume-fs

Purpose

List the file shares in your organization.

Example

```
bx ic volume-fs
```

bx ic volume-fs-remove FSNAME

Purpose

Remove a file share.

Parameters

FSNAME

(Required) The file share name.

Example

```
bx ic volume-fs-remove my_fileshare
```

bx ic volume-inspect VOLNAME

Purpose

Inspect the volume.

Parameters

VOLNAME

(Required) The volume name.

Example

The following example is a request to inspect the volume, where *my_volume* is the name of the volume.

```
bx ic volume-inspect my_volume
```

bx ic volume-remove VOLNAME

Purpose

Remove the volume.

Parameters

VOLNAME

(Required) The volume name.

Example

The following example is a request to remove the volume, where *my_volume* is the name of the volume.

```
bx ic volume-remove my_volume
```

bx ic volumes [--table]

Purpose

List the volumes.

Parameters

--table

(Optional) Structures the list output in a table format.

Example

The following example is a request to list all the volumes.

```
bx ic volumes [--table]
```

bx ic wait CONTAINER [CONTAINER]

Purpose

Wait for a container to exit. During this waiting time your command line does not return and you cannot enter commands. As soon as the container exits or stops, the exit code is printed in the CLI output. For more information, see the [wait](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID.

Example

The following example is a request to wait for a container that is called *my_container* to exit. As soon as it exits, the exit code is printed in the CLI.

```
bx ic wait my_container
```

bx ic wait-status CONTAINER

Purpose

Wait for a single container or container group to reach a non-transient state. During this waiting time your command line does not return and you cannot enter commands. As soon as the container reaches a non-transient state, an **OK** message is displayed. For single containers, the non-transient states include Running, Shutdown, Crashed, Paused, or Suspended. For container groups, the non-transient states include CREATE_COMPLETE, UPDATE_COMPLETE, or FAILED.

Parameters

CONTAINER

(Required) The container or group name or ID.

Example

The following example is a request to wait for a container that is called *my_container* until it reaches a non-transient state.

```
bx ic wait-status my_container
```

IBM Cloud Container Service plug-in (bx ic) command quick reference

Table 8. *bx ic* group command quick reference

Container groups
<p>Group processes</p> <pre>bx ic group-create [--anti] [--auto] [-d DOMAIN] [--desired DESIRED] [-e ENV] [--env-file ENVFILE] [--http_monitor_enabled] [--http_monitor_path] [--http_monitor_rc_list] [--ip IP_ADDRESS] [-m MEMORY] [--max MAX] [--min MIN] [-n HOST] --name NAME [-p PORT] [--session-affinity] [--volume VOLUME:/DIRECTORY_PATH] IMAGE [CMD]</pre> <pre>bx ic group-remove [-f] GROUP [GROUP]</pre> <pre>bx ic group-update [--desired DESIRED] [-e "KEY=VALUE"] [--max MAX] [--min MIN] GROUP</pre> <pre>bx ic port CONTAINER [PRIVATE_PORT[/PROTOCOL]]</pre>
<p>Information gathering</p> <pre>bx ic group-inspect GROUP</pre> <pre>bx ic group-instances GROUP</pre> <pre>bx ic groups [-q]</pre> <pre>bx ic ps [-a] [--filter env=SEARCH_CRITERIA] [-l NUM] [-q] [-s]</pre>
<p>Instance management</p> <pre>bx ic attach [--no-stdin] [--sig-proxy] CONTAINER</pre> <pre>bx ic exec [-d] [-it] [-u USER] CONTAINER CMD</pre> <pre>bx ic inspect [IMAGE images CONTAINER]</pre> <pre>bx ic kill [-s CMD] CONTAINER [CONTAINER]</pre> <pre>bx ic logs [-f --since TIMESTAMP --tail STRING -t] CONTAINER</pre> <pre>bx ic pause CONTAINER [CONTAINER]</pre> <pre>bx ic port CONTAINER [PRIVATE_PORT[/PROTOCOL]]</pre>

Container groups

```
bx ic restart CONTAINER [CONTAINER] [-t SECS]
bx ic rename OLD_NAME NEW_NAME
bx ic rm [-f] CONTAINER [CONTAINER]
bx ic start CONTAINER [CONTAINER]
bx ic stats --no-stream CONTAINER [CONTAINER]
bx ic stop CONTAINER [CONTAINER] [-t SECS]
bx ic top CONTAINER [CONTAINER] [ps options]
bx ic unpause CONTAINER [CONTAINER]
bx ic wait CONTAINER [CONTAINER]
bx ic wait-status CONTAINER
```

Routes

```
bx ic route-map [-d DOMAIN] [-n HOST] GROUP
bx ic route-unmap [-d DOMAIN] [-n HOST] GROUP
```

Service binding

```
bx ic service-bind GROUP SERVICE_INSTANCE
bx ic service-unbind GROUP SERVICE_INSTANCE
```

Table 9. *bx ic* single container command quick reference

Single containers

Processes

```
bx ic attach [--no-stdin] [--sig-proxy] CONTAINER
bx ic exec [-d] [-it] [-u USER] CONTAINER CMD
bx ic kill [-s CMD] CONTAINER [CONTAINER]
bx ic rename OLD_NAME NEW_NAME
bx ic rm [-f] CONTAINER [CONTAINER]
bx ic run [-p PORT] [-P] [-d] [-e ENV] [--env-file ENVFILE] [-it] [--link NAME:ALIAS] [-m MEMORY] --name NAME [--volume VOLUME:/DIRECTORY_PATH] IMAGE [CMD [CMD ...]]
```

Information gathering

```
bx ic attach [--no-stdin] [--sig-proxy] CONTAINER
bx ic inspect [IMAGE | images | CONTAINER]
```

Single containers

```
bx ic logs [-f | --since TIMESTAMP | --tail STRING | -t] CONTAINER
bx ic port CONTAINER [PRIVATE_PORT[/PROTOCOL]]
bx ic ps [-a] [--filter env=SEARCH_CRITERIA] [-l NUM] [-q] [-s]
bx ic stats --no-stream CONTAINER [CONTAINER]
bx ic top CONTAINER [CONTAINER] [ps options]
```

IP addresses

```
bx ic ip-bind ADDRESS CONTAINER

bx ic ip-release ADDRESS
bx ic ip-request [-q]
bx ic ip-unbind ADDRESS CONTAINER
bx ic ips [-q]
```

Status changes

```
bx ic pause CONTAINER [CONTAINER]
bx ic restart CONTAINER [CONTAINER] [-t SECS]
bx ic start CONTAINER [CONTAINER]
bx ic stop CONTAINER [CONTAINER] [-t SECS]
bx ic unpause CONTAINER [CONTAINER]
bx ic wait CONTAINER [CONTAINER]
bx ic wait-status CONTAINER
```

Table 10. *bx ic* image command quick reference

Images

```
bx ic build -t TAG [--no-cache] [--pull] [-q] DOCKERFILE_LOCATION
bx ic cp SRC_PATH CONTAINER:DEST_PATH | bx ic cp CONTAINER:SRC_PATH DEST_PATH
bx ic cpi SRC DST
bx ic images [-a] [-f CONDITION] [--no-trunc] [q]
bx ic inspect [IMAGE | images | CONTAINER]
bx ic namespace-[get | set] NAME
bx ic rmi [-R REGISTRY] IMAGE [IMAGE]
```

Table 11. `bx ic` storage command quick reference

Data storage
File shares
<code>bx ic volume-fs</code> <code>bx ic volume-fs-create FSNAME FSSIZE FSIOPS</code>
<code>bx ic volume-fs-flavors</code>
<code>bx ic volume-fs-inspect FSNAME</code>
<code>bx ic volume-fs-remove FSNAME</code>
Volumes
<code>bx ic volume-create VOLNAME [FSNAME]</code>
<code>bx ic volume-inspect VOLNAME</code>
<code>bx ic volume-remove VOLNAME</code>
<code>bx ic volumes [--table]</code>

Table 12. `bx ic` CLI and service management command quick reference

CLI and service management
<code>bx ic info [--json]</code>
<code>bx ic init</code>
<code>bx ic reprovision [-f] [AVAILABILITY_ZONE]</code>
<code>bx ic unprovision [-f]</code>
<code>bx ic update</code>
<code>bx ic version</code>

Version history for the IBM Cloud Container Service plug-in (`bx ic`)

Review the changes associated with each version of the IBM Cloud Container Service plug-in (`bx ic`). You might be prompted to update the plug-in periodically to use new features.

Tip

To see which version is installed, run the `bx plugin list` command.

Table 13. IBM Cloud Container Service plug-in version history

Release date	Version	Updates
November 16, 2016	0.8.964	<ul style="list-style-type: none">Added support for managing the health check for container groups through the <code>bx ic group-create</code> command by using the <code>--http_monitor_enabled</code>, <code>--http_monitor_path</code>, and <code>--http_monitor_rs_list</code> optionsAdded support for the <code>bx ic group-create --session_affinity</code> optionDefect fixes
September 23, 2016	0.8.951	<ul style="list-style-type: none">Added support for <code>--ip</code> option in the <code>bx ic group-create</code> command that binds a public floating IP address to the container groupDefect fixes
August 24, 2016	0.8.934	<ul style="list-style-type: none">Improved error messaging for missing or incorrect command optionsAdded support for the <code>--json</code> option with <code>bx ic info</code>Added support for anti-affinity (<code>--anti</code> option) in the <code>bx ic group-create</code> command that forces every container instance in a container group to be placed on a separate physical compute node
June 28, 2016	0.8.897	<ul style="list-style-type: none">Added support for the <code>-q</code> option with <code>bx ic ips</code> and <code>bx ic groups</code>Changed the time that displays with <code>bx ic groups</code> from UTC

Release date	Version	Updates
		<p>time to the local time</p> <ul style="list-style-type: none"> Added support for removing more than one IP address at a time with <code>bx ic ip-remove</code> Defect fixes
May 17, 2016	0.8.889	<ul style="list-style-type: none"> Added support for <code>bx ic cp</code> Behavior updates for <code>bx ic login</code> and <code>bx ic init</code>. If a certificate is already available on the system, <code>bx ic login</code> uses the existing certificate. <code>bx ic init</code> always requests a new certificate. Defect fixes
May 3, 2016	0.8.878	<ul style="list-style-type: none"> Added support for <code>bx ic group-create --env-file</code> Defect fixes
April 15, 2016	0.8.876	<ul style="list-style-type: none"> Added support for <code>bx ic reprovision</code> and <code>bx ic unprovision</code> Added volumes support for Docker Compose
April 8, 2016	0.8.873	<ul style="list-style-type: none"> Defect fix for <code>bx ic volume-fs-create</code>
March 31, 2016	0.8.868	<ul style="list-style-type: none"> Added a spinner for long-running commands Added support for <code>bx ic volumes --table</code> Added support for <code>bx ic wait-status <container_id></code> Defect fixes
18 February 2016	0.8.826	<ul style="list-style-type: none"> Added support for the <code>rename</code> native Docker command Added support for <code>docker compose</code>

Release date	Version	Updates
12 February 2016	0.8.823	<ul style="list-style-type: none"> • Added file system commands for volumes • During login, Docker QuickStart Terminal was added as part of Option 2 for setting environment variables to use Docker commands • Improved messaging
13 November 2015	0.8.788	<ul style="list-style-type: none"> • Capability to switch regions by running <code>cf login -a</code> without logging into <code>bx ic</code> again • Automatic refresh the <code>cf</code> token • Improved messaging • Added information in <code>bx ic info</code> • Defect fixes
26 October 2015	0.8.763	<ul style="list-style-type: none"> • Visual design improvements in the CLI • Added support for the <code>exec</code>, <code>attach</code>, <code>logs -f</code>, <code>kill</code>, <code>stats</code>, and <code>run -it</code> native Docker commands • Use <code>cf login</code> to switch regions without logging into <code>bx ic</code>.
2 October 2015	0.8.723	<ul style="list-style-type: none"> • Using <code>-n</code> to specify a name with <code>bx ic group-create</code> is deprecated • Added support for <code>tls_key</code> authentication with the <code>rmi</code> native Docker command
16 September 2015	0.8.716	<ul style="list-style-type: none"> • Message updates for <code>bx ic route-map</code> and <code>bx ic route-unmap</code> • Fixed <code>bx ic group-create</code> defects • Fixed <code>bx ic version</code> display issues

Release date	Version	Updates
3 September 2015	0.8.704	<ul style="list-style-type: none"> • Expanded information for <code>bx ic inspect images</code> • Fixed <code>bx ic cpi</code> defect • Added support for <code>bx ic run -P</code> to automatically expose ports in your container that are specified in the Dockerfile, which can be used in place of <code>-p <port></code> to indicate individual ports • Added support for <code>bx ic group-remove -f</code> to forcibly delete running or failed containers. • Added support for <code>bx ic group-create [-n --hostname] [-d --domain]</code> to specify the host and domain for a container group route. • Fixed defect where only the first of multiple environment variables (<code>-e, --env</code>) specified with <code>bx ic group-create</code> is used. • Running <code>bx ic version</code> and <code>docker version</code> now includes version information for the IBM Cloud Container Service API
20 July 2015	0.8.646	<ul style="list-style-type: none"> • Added <code>bx ic update</code> command so that the latest available plug-in version can be checked against the installed plug-in version • Fixed Windows defect for the <code>cf</code> version check
17 July 2015	0.8.640	<ul style="list-style-type: none"> • Wildcard matching for container and image IDs • For the <code>images</code> command, Docker image ID is displayed instead the Open Stack ID • Enabled debug mode • Enabled CF version verification • Improved performance of <code>bx ic info</code> • Added support for the <code>build native Docker</code> command • Added support for the <code>inspect images native Docker</code> command

Release date	Version	Updates
		<ul style="list-style-type: none"> You can use <code>bx ic group-create --desired NUMBER</code> to set the number of instances to create, otherwise the number that is specified with <code>group-create --max NUMBER</code> is the number of instances that are created

Container states in IBM Cloud Container Service

You can view the current container state by running the `bx ic inspect` command and locating the **Status** field. The container state can give you additional information about the availability of the container instances and problems that might have occurred.

Single container states

Table 14. Single container states in IBM Cloud Container Service

Status	Description	Tip
BUILDING	The creation of the container is in progress.	The creation of the container can take a few minutes to complete.
CRASHED	The app or the container exited with an error, resulting in a crashed status.	Run <code>bx ic logs <container></code> and look at the log files for the container to see why the app exited with an error. Assure that IBM Cloud Container Service networking is finished before the app starts by adding a sleep command to your Dockerfile .
NETWORKING	If the container is exposed to the public, the network connection is set up for the container during creation.	Establishing the network connection for the container can take a few minutes to complete.
PAUSED	All processes that are running inside the container are stopped. You cannot	Unpause the container to restart the processes inside the container.

Status	Description	Tip
	access the app inside the container until you unpause the container.	
RUNNING	The container is up and running.	You can access your app by either using the assigned private IP addresses, or, if exposed to the public, by using the bound public IP address.
SHUTDOWN	The container was either stopped or the process inside the container ended.	Make sure to use a long-running process to keep your container up and running. If the container was stopped, you can start it again.

Container group states

Table 15. Container group states in IBM Cloud Container Service

Status	Description	Tip
CONFIGURING_NETWORK	When you exposed the container group to the public, the public route is registered with the GoRouter and the group load balancer is set up to allow incoming public data on the exposed public port.	The network configuration can take a few minutes to complete. If the state does not change, try to re-create the container group. If this state persists after re-creating the group, open a Cloud support ticket .
CREATE_COMPLETE	The container group is up and running.	You can access your app by either using the assigned private IP address of the group load balancer, or, if exposed to the public, by using the route.
CREATE_FAILED	The container group could not be created	Run <code>bx ic group-inspect GROUP</code> to view the root cause of the group creation failure. If the

Status	Description	Tip
	because the app exited with an error, or due to an issue in OpenStack.	error is returned from the app, run <code>bx ic group-instances GROUP</code> to retrieve the ID of the group instances. Then, run <code>bx ic logs CONTAINER</code> for logging information from the app or <code>bx ic inspect CONTAINER</code> for container specific error information. If the error occurred due to an issue in OpenStack, wait a few minutes and try re-creating the container group. If this state persists, open a Cloud support ticket .
CREATE_IN_PROGRESS	The creation of the container group is in progress.	The creation of the container group can take a few minutes to complete. When completed successfully, the status CREATE_COMPLETE is displayed. If the creation fails, CREATE_FAILED is displayed.
DELETE_FAILED	The container group could not be deleted due to an issue in OpenStack.	Wait a few minutes and then try again. If this state persists, open a Cloud support ticket .
DELETE_IN_PROGRESS	The deletion of the container group was requested.	The deletion of the container group and all its instances can take a few minutes. When the container group is removed, the group disappears from the container list and the assigned quota is released.
HAS_INACTIVE_INSTANCES	One or more instances of the container group lost or does not yet have network connection, and display an INACTIVE status. Incoming data	When you create your container group, make sure to enable the auto-recovery mode by using the <code>--auto</code> option to automatically replace the unresponsive instance with a new instance and resolve the HAS_INACTIVE_INSTANCES status. If auto-recovery is not enabled, you must re-create the container group to resolve this state.

Status	Description	Tip
	is routed to the remaining instances that are marked as ACTIVE .	Also, when you create your container group, if the app requires a port to be exposed, make sure to include that port number with the -p option.
NO_ACTIVE_INSTANCES	If no active instances are left, then your container group becomes unavailable and you cannot access it by using the private IP address of the group load balancer or route.	If this state persists, networking might not be configurable for the app when it is deployed as a group. Try deploying the app as a single container instead.
UPDATE_COMPLETE	The update was completed successfully.	When group instances were added or removed, the quota usage is updated automatically.
UPDATE_FAILED	The container group could not be updated due to an issue in OpenStack.	Wait a few minutes and then try again. If this state persists, open a Cloud support ticket .
UPDATE_IN_PROGRESS	This status is displayed when an update of the container group was requested by using the <code>bx ic group-update</code> command, or, when auto-recovery is enabled, to recover an unresponsive group instance.	The update of the container group can take a few minutes to complete. When completed successfully, the status UPDATE_COMPLETE is displayed. If the update fails, UPDATE_FAILED is displayed.

Supported Docker commands for IBM Cloud Container Service plug-in (bx ic)

With the IBM Cloud Container Service plug-in (bx ic), some docker commands are supported for use in place of bx ic commands. When you log in to the plug-in, configure the environment variables to use the Docker CLI commands. Then, you can run docker commands instead of bx ic commands whenever possible. For example, instead of using bx ic images to see the images in your private Cloud registry, run docker images.

Note

Some bx ic commands, like bx ic namespace-get do not have a docker alternative. For a complete list of bx ic commands, see [CLI reference for managing single and scalable containers in IBM Cloud Container Service \(Deprecated\)](#).

Table 16. Supported Docker commands

Docker command	Supported for use as bx ic <command>	Notes	Available documentation in IBM Cloud Container Service
attach	Supported	Supported options: --no-stdin Unsupported options: --help	
build	Supported	Supported options: --force-rm, --no-cache, --pull, -t, --tag	Learn more about how to use bx ic build

Docker command	Supported for use as <code>bx ic <command></code>	Notes	Available documentation in IBM Cloud Container Service
		Unsupported options: -f, --file, --help, -q, --quiet	
commit	Not supported		
cp	Supported	Unsupported options: --help	
create	Not supported		
diff	Not supported		
events	Not supported		
exec	Supported	Supported options: -d , --detach, -i, --interactive, -t Unsupported options: --help	Learn more about how to use <code>bx ic exec</code>
export	Not supported		
history	Not supported		
images	Supported	Supported options: -a , --all, --no-trunc , -q, --quiet Unsupported options: --digests, -f, --filter, --help	
import	Not supported		

Docker command	Supported for use as <code>bx ic <command></code>	Notes	Available documentation in IBM Cloud Container Service
inspect	Supported	Supported options: -f , --format, --type Unsupported options: --help, -s, --size	
kill	Supported	Supported options: -s Unsupported options: --help	
load	Not supported		
login	Not supported		
logout	Not supported		
logs	Supported	Supported options: -f , --follow, --since , -t, --timestamps, --tail Unsupported options: --help	
network connect	Not supported		
network create	Not supported		
network disconnect	Not supported		
network inspect	Not supported		
network ls	Not supported		
network rm	Not supported		

Docker command	Supported for use as <code>bx ic <command></code>	Notes	Available documentation in IBM Cloud Container Service
pause	Supported	Unsupported options: --help	
port	Supported		
ps	Supported	Supported options: -a, --all, --format, --latest, --no-trunc, -q, --quiet, -s, --size Unsupported options: -f, --filter, --help, -l, -n	
pull	Not supported		Learn more about how to use docker pull with IBM Cloud Container Service instead
push	Not supported		Learn more about how to use docker push with IBM Cloud Container Service instead
rename	Supported	Unsupported options: --help	
restart	Supported	Supported options: -t, --time Unsupported options: --help	

Docker command	Supported for use as <code>bx ic <command></code>	Notes	Available documentation in IBM Cloud Container Service
rm	Supported	Supported options: -f, --force Unsupported options: --help	Learn more about how to use <code>bx ic rm</code>
rmi	Supported	Supported options: -f, --force, --no-prune Unsupported options: --help	Learn more about how to use <code>bx ic rmi</code>.
run	Supported	Required options: -d Supported options: -a, --attach, -e, --detached, --env, --env-file, --expose, --link, -m, --memory, --name, -p, --publish, --publish-all, --volume Unsupported options: --privileged, --dns, --dns-opt, --dns-search, -h, --help, --hostname, --help, -i,	Learn more about how to use <code>bx ic run</code> <div> Tip For the -m option, when no unit of measurement is specified with the value, bytes is used. </div>

Docker command	Supported for use as bx ic <command>	Notes	Available documentation in IBM Cloud Container Service
		--interactive, -it , --log-driver, --log-opt, -t, -v, --volumes-from, -w , --workdir	
save	Not supported		
search	Not supported		
start	Supported	Supported options: -a , --attach, -i, --interactive Unsupported options: --help	
stats	Supported	Supported options: --no-stream Unsupported options: --help	
stop	Supported	Unsupported options: --help, -t, --time	
tag	Not supported		
top	Supported	Unsupported options: --help	
unpause	Supported	Unsupported options: --help	
version	Supported	Supported options: -f , --format	

Docker command	Supported for use as <code>bx ic <command></code>	Notes	Available documentation in IBM Cloud Container Service
		Unsupported options: <code>--help</code>	
<code>wait</code>	Supported	Unsupported options: <code>--help</code>	

Docker Compose commands for managing multi-container deployments with the IBM Cloud Container Service plug-in (`bx ic`)

Refer to these commands to run and manage your multi-container deployment with the IBM Cloud Container Service plug-in (`bx ic`). When you [log in to the plug-in](#), configure the environment variables for **Option 2** to use the Docker CLI commands. Then, you can execute Docker Compose commands to manage your multi-container deployment. Be sure to be in the directory where your `docker-compose.yml` file is located before you execute Docker Compose commands. Otherwise, an error message is displayed.

Table 17. Commands for using Docker Compose on Cloud

Commands for using Docker Compose on Cloud				
<code>docker-compose logs</code>	<code>docker-compose ps [-q]</code>	<code>docker-compose rm [SERVICE]</code>	<code>docker-compose scale SERVICE=NUM</code>	<code>docker-compose start [SERVICE]</code>
<code>docker-compose stop [-t] [SERVICE]</code>	<code>docker-compose up [-d]</code>			

`docker-compose logs`

Purpose

This command shows log information for each service of your multi-container deployment.

Note

This command is currently not supported by IBM Cloud Container Service.

docker-compose ps [-q]

Purpose

This command lists all running and stopped containers that are part of your container system. For more information, see the [docker-compose ps](#) command in the Docker help.

Supported command options

-q

(Optional) Use the -q option if you want to display container IDs only.

Example

The following example is a request to list all container instances in a container system.

```
docker-compose ps
```

docker-compose rm [SERVICE]

Purpose

If you specify a service, this command removes the container instance that runs that service. If you do not specify a service, the whole container system is removed. In both cases, it is important that the service is stopped before you remove the container, otherwise an error is displayed. For more information, see the [docker-compose rm](#) command in the Docker help.

Parameters

SERVICE

(Optional) The name of the container service you want to remove.

Note

The name of the service can be found in the `docker-compose.yml` file. Do not use the name of the container that runs the service, which was created when you ran the `docker-compose up` command.

Example

The following example is a request to remove the container that runs the service named *my_service*.

```
docker-compose rm my_service
```

docker-compose scale SERVICE=NUM

Purpose

Use this command to increase or decrease the number of container instances that run a service. It is recommended to have more than one container instance if your service requires scalability and reliability. For more information, see the [docker-compose rm](#) command in the Docker help.

Parameters

SERVICE

(Required) The name of the service for which you want to increase or decrease the number of containers.

Note

The name of the service can be found in the `docker-compose.yml` file. Do not use the name of the container that runs the service, which was created when you ran the `docker-compose up` command.

NUM

(Required) The number of containers you want to create for your service.

Example

The following example is a request to create 3 container instances for a service that is named *my_service*.

```
docker-compose scale my_service=3
```

docker-compose start [SERVICE]

Purpose

If you specify a service, this command starts all container instances that are linked to that service. If you do not specify a service, all container instances in your container system are started.

Note

This command can be used only if container instances were already created by using the `docker-compose up` command.

For more information, see the [docker-compose start](#) command in the Docker help.

Parameters

SERVICE

(Optional) The name of the service you want to start.

Note

The name of the service can be found in the `docker-compose.yml` file. Do not use the name of the container that runs the service, which was created when you ran the `docker-compose up` command.

Example

The following example is a request to start a service that is named *my_service*.

```
docker-compose start my_service
```

docker-compose stop [-t] [SERVICE]

Purpose

If you specify a service, this command stops all container instances that run that service. If you do not specify a service, all container instances in your container system are stopped. Container services that are stopped are not removed, so they can be started again. For more information, see the [docker-compose stop](#) command in the Docker help.

Parameters

SERVICE

(Optional) The name of the running service that you want to stop.

Note

The name of the service can be found in the `docker-compose.yml` file. Do not use the name of the container that runs the service, which was created when you ran the `docker-compose up` command.

Supported command options

-t, --timeout

(Optional) Use the `-t` or `--timeout` option to change the default shutdown timeout. A shutdown timeout is the time the Docker Daemon has to wait for the container instance to respond that it is stopping. The default shutdown timeout is 10 seconds.

Example

The following example is a request to stop a service that is named *my_service*.

```
docker-compose stop my_service
```

docker-compose up [-d]

Purpose

This command builds, (re)creates, starts, and links the container instances for the services you defined in your `docker-compose.yml` file. If your multi-container deployment is already running, this command checks for updates you defined in your `docker-compose.yml` file and stops and re-creates only those containers where the configuration changed. For more information, see the [docker-compose up](#) command in the Docker help.

Supported command options

-d

(Optional) You can either run the `docker-compose up` command interactively, so you can see all steps during container creation, or use the `-d` option to start and run your containers in the background. In IBM Cloud Container Service, most services and apps are long running tasks. Because of this, leverage the `-d` option to avoid getting stuck in the console output of your containers with no way to detach. If you use the `-d` option, container names are still displayed in the CLI after they are created.

Example

The following example is a request to create your multi-container deployment and its dependencies.

```
docker-compose up
```

Supported CLIs

IBM Cloud Container Service plug-in

With the current version of the IBM Cloud Container Service plug-in, the versions of these prerequisites are supported.

Docker version 1.8 to 1.12

Cloud Foundry CLI version 6.14.0 - 6.22.0

Note

MacOS Sierra users must install Cloud Foundry CLI version 6.22.0 or later.

Docker Compose

If you choose to run Docker Compose commands with IBM Cloud Container Service, the versions of these prerequisites are supported.

Docker version 1.10.0

Cloud Foundry CLI version 6.14.0 - 6.22.0

IBM Cloud Container Service plug-in version 0.8.826 or later

Docker Compose version 1.6.0 or later

(Deprecated) IBM Cloud Container Service plug-in (cf ic) commands for managing containers

With the IBM Cloud Container Service Cloud Foundry plug-in, refer to these commands to create and manage containers.

You can run any of these commands without any options to review help information. You can also run `cf ic --help` to view a list of these commands.

The cf ic CLI has been deprecated

Instead of using the IBM Cloud Container Service plug-in for Cloud Foundry (`cf ic`), use the IBM Cloud Container Service plug-in for Cloud. You can continue to use the `cf ic` CLI until 01 January 2018.

To start using the IBM Cloud Container Service plug-in for Cloud instead, refer to the following topics.

[Installing the CLI](#)

[Installing the bx ic plug-in](#)

Table 18. Commands for managing the CLI and your service on Cloud

Commands for managing the CLI and your service on Cloud			
cf ic info	cf ic init	cf ic login	cf ic namespace
cf ic reprovision	cf ic unprovision	cf ic update	cf ic version

Table 19. Commands for creating and managing containers on Cloud

Commands for creating and managing containers on Cloud				
cf ic attach	cf ic bind-service	cf ic cp	cf ic exec	cf ic group create
cf ic group inspect	cf ic group instances	cf ic group list [-q]	cf ic group rm	cf ic group update
cf ic inspect	cf ic ip bind	cf ic ip list	cf ic ip release	cf ic ip request
cf ic ip unbind	cf ic kill	cf ic logs	cf ic pause	cf ic port
cf ic ps	cf ic rename	cf ic restart	cf ic rm	cf ic route map
cf ic route unmap	cf ic run	cf ic start	cf ic stats	cf ic stop
cf ic top	cf ic unbind-service	cf ic unpause	cf ic wait	cf ic wait-status

Table 20. Commands for managing images on Cloud

Commands for managing images on Cloud				
cf ic build	cf ic cpi	cf ic images	cf ic inspect	cf ic rmi

Table 21. Commands for storage for containers on Cloud

Commands for storage for containers on Cloud
--

Commands for storage for containers on Cloud

cf ic volume create	cf ic volume fs-create	cf ic volume fs-flavor-list	cf ic volume fs-inspect	cf ic volume fs-list
cf ic volume fs-rm	cf ic volume inspect	cf ic volume list	cf ic volume rm	

cf ic attach [--no-stdin] [--sig-proxy] CONTAINER

Purpose

Control a running container or view its output. Use CTRL+C to exit and stop the container. For more information, see the [attach](#) command in the Docker help.

Parameters

--no-stdin

(Optional) Do not include the standard input.

--sig-proxy

(Optional) The default is true.

CONTAINER

(Required) A container name or ID.

Example

The following example is a request to attach to the container *my_container*.

```
cf ic attach my_container
```

cf ic bind-service GROUP SERVICE_INSTANCE

Purpose

Add a service to a running container group. This command is only available to container groups. Single containers must bind a service as part of the `cf ic run` command. To unbind a service, see the [unbind-service](#) command.

Parameters

GROUP

(Required) The group ID or name.

SERVICE_INSTANCE

(Required) The name of the service instance to be added to the container group.

Example

The following example is a request to bind an instance of the Data Cache service named MyDataCache to the container *my_container_group*.

```
cf ic bind-service my_container_group MyDataCache
```

cf ic build -t TAG [--no-cache] [--pull] [-q] DOCKERFILE_LOCATION

Purpose

Build a Docker image locally or in the private Cloud images registry. For more information, see the [build](#) command in the Docker help.

Parameters

DOCKERFILE_LOCATION

(Required) The path to the Dockerfile and context on the local host.

--no-cache

(Optional) When an image is built for the first time, the layers for the image are cached for an entire organization. When you build subsequent versions of the image, these cached versions of the image are used to speed up the build process. By including the --no-cache option when you build a new version of the image, you can ensure that each layer of the image is re-built too.

--pull

(Optional) By default, the base image is only pulled from the registry if it isn't already in your organization's image cache. By including this option, you can ensure that latest version of the base image is used to build the new image.

-q, --quiet

(Optional) Suppress the verbose output that is generated by the containers. The default is false.

-t TAG, --tag TAG

(Required) The path to your private images registry to apply to the image

that is created.

Example

The following example is a request to build an image that is named `myimage`. The `Dockerfile` and other artifacts to use in the build are in the same directory that the command is run from. Because the registry and namespace are included with the image name, the image is built in your organization's private Cloud images registry.

Tip

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

```
cf ic build -t registry.DomainName/<my_namespace>/myimage .
```

cf ic cp SRC_PATH CONTAINER:DEST_PATH | cf ic cp CONTAINER:SRC_PATH DEST_PATH

Purpose

Copy a file or folder from a running or stopped single container to your local machine and vice versa. For more information, see the [cp](#) command in the Docker help.

Note

This command is supported to be used with single containers only. When you want to copy files from or to a container group, you must copy the files from or to each container instance of the group.

Parameters

DEST_PATH

(Required) The path to the file or folder on your local machine or container where you want to copy the information from the source path. The file or folder must exist on your local machine or the container before you can

start copying. If you want to copy to your local machine, you can use the absolute or relative path. If you want to copy to your single container, you must include the name of the container, the delimiter ":", and the path to the destination file or folder. The path must be relative to the root directory of the container.

SRC_PATH

(Required) The path to the file or folder that you want to copy. If the file or folder is on your local machine, you can use the absolute or relative path. If the file or folder is in your single container, you must include the name of the container, the delimiter ":", and the path to the file or folder that you want to copy. The path must be relative to the root directory of the container.

Note

You cannot copy certain system files that are stored under the /dev, /etc, /proc and /sys folder of your container.

Example

This example command copies a file that is named `config.txt` from the current directory of your local machine to the `config` directory of a single container that is named *my_container*.

```
cf ic cp ./config.txt my_container:/config
```

cf ic cpi SRC DST

Purpose

Access a Docker Hub image or an image from your local registry and copy it to your private Cloud images registry.

Note

This command is an alternative to using `docker pull` and `docker push` together to add an image to your Cloud images registry. If you prefer to use `docker pull` and `docker push`, see [Pushing local images to your private Cloud registry from the command line](#).

Parameters

DST

(Required) The private Cloud images registry URL, which includes the namespace, and the destination image name. A tag for the image is optional.

SRC

(Required) The source repository and image name.

Example

```
cf ic cpi source_repository/source_image_name private_registry_URL/d
```

Tip

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

```
cf ic cpi training/sinatra registry.DomainName/<my_namespace>/mysina
```

cf ic exec [-d] [-it] [-u USER] CONTAINER CMD

Purpose

Execute a command within a container. For more information, see the [exec](#) command in the Docker help.

Parameters

CMD

(Required) The command to execute within the container or containers

specified.

CONTAINER

(Required) A container name or ID.

-d , --detach

(Optional) Run the specified command in the background.

-it

(Optional) Interactive mode. Keep the standard input display. Type `exit` to exit.

-u USER, --user USER

(Optional) A user name.

Examples

This example command runs an interactive bash terminal in the container *my_container*.

```
cf ic exec -it my_container bash
```

This example command runs the `date` command in the container *my_container*.

```
cf ic exec my_container date
```

cf ic group create [--anti] [--auto] [-d DOMAIN] [--desired DESIRED] [-e ENV] [--env-file ENVFILE] [--http_monitor_enabled] [--http_monitor_path] [--http_monitor_rc_list] [--ip IP_ADDRESS] [-m MEMORY] [--max MAX] [--min MIN] [-n HOST] --name NAME [-p PORT] [--session_affinity] [--volume VOLUME:/DIRECTORY_PATH] IMAGE [CMD]

Purpose

Create a scaling group. By default, containers in Cloud run in detached mode.

Parameters

--anti

(Optional) Use anti-affinity to make your container group more highly available. The `--anti` option forces every container instance in your group to be placed on a separate physical compute node, which reduces the odds

of all containers in a group crashing due to a hardware failure. You might not be able to use this option with larger group sizes because each Cloud region and organization has a limited set of compute nodes available for deployment. If your deployment does not succeed, either reduce the number of container instances in the group or remove the `--anti` option.

--auto

(Optional) When automatic recovery is enabled for a container group and the group has been running for 10 minutes, the group's load balancer starts to regularly check the health of each container instance in the group via HTTP requests. If a container instance does not respond within 100 seconds, it is marked as inactive. Inactive container instances are removed from the group and re-created by auto-recovery. Auto-recovery attempts to recover container instances in a group 3 times. After the third attempt, auto-recovery does not recover any container instances in the group for 60 minutes. After 60 minutes, the auto-recovery process starts again. For more information about the health check, see the `--http_monitor_enabled` option.

CMD

(Optional) The command and arguments are passed to the container group to execute. This command must be a long-running command. Do not use a short-lived command, for example, **/bin/date**, because it might cause the container to crash.

Example long-running commands using the `ibmnode` image that is provided by IBM Cloud Container Service:

```
cf ic group create --name my_container_group registry.Domain
```

```
cf ic group create --name my_container_group registry.Domain
```

Note

The CMD and its arguments must come at the end of the `cf ic group create` command line.

-d DOMAIN, --domain DOMAIN

(Optional) The default system domain is `AppDomainName` and already provides a SSL certificate, so you can access your container groups with HTTPS without any additional configuration. To use a custom domain, you must register the custom domain on a public DNS server, configure the custom domain in Cloud, and then map the custom domain to the Cloud system domain on the public DNS server. After your custom domain is mapped to the Cloud system domain, requests for your custom domain are routed to your application in Cloud. When you create a custom domain, do not include underscores (`_`) in the domain name. For more information about custom domains, see [Creating and using a custom domain](#). To make your custom domain secure, [upload a SSL certificate](#), so your container groups can be accessed with HTTPS.

The host and the domain combined form the full public route URL, such as `http://mycontainerhost.AppDomainName` and must be unique within Cloud. When you review the details of a container group with the `cf ic group inspect` command, the host and the domain are listed together as the route.

--desired DESIRED

(Optional) The number of instances in your group. The default is 2. The number of container group instances that your app requires depends on the expected workload and the time in which a request can be processed by the app. For example, multi-tasking or network connectivity affects how many instances might be required for your app. To determine the number of desired instances, deploy a single container in Cloud that runs your app and perform a load test on this container. If the app can handle, for example, 100 requests per second, but you are expecting an average workload of 300 requests per second, then the desired number of instances for the container group is 3. To make your container group highly available and more resilient to failure, consider including extra instances than the minimum to handle the expected workload. Extra instances can handle the workload in case another instance crashes. To ensure automatic recovery of a failed instance without affecting the workload, include one extra instance. For protection against two simultaneous failures, include two extra instances. This set up is an $N+2$ pattern, where N is the number of instances

to handle the requests and +2 is an extra two instances.

Important

A container group should include at least 2 container instances. If you include only 1 instance in the group and that instance becomes unavailable, your users can experience downtime for the app.

-e ENV, --env ENV

(Optional) Set the environment variable where ENV is a key=value pair. List multiple keys separately and if you include quotation marks, include them around both the environment variable name and the value. Example:

```
-e "key1=value1" -e "key2=value2a,value2b" -e "key3=value3"
```

Table 22. Suggested environment variables

Environment variable	Description
CCS_BIND_APP= <appname>	Some Cloud services do not support direct binding to a container. In this case, you need to create a Cloud Foundry app and bind the Cloud service to it. Then, you bind the app to your container by using CCS_BIND_APP. The Cloud Foundry app acts as a bridge and allows Cloud to inject your bridge app's VCAP_SERVICES information into the running container instance.
CCS_BIND_SRV= <service_instance_name1>, <service_instance_name2>	To bind a Cloud service directly to a container without using a bridge app, use CCS_BIND_SRV. This binding allows Cloud to inject the VCAP_SERVICES information into

Environment variable	Description
<div>Note</div> <p>When a service does not support the use of the CCS_BIND_SRV= environment variable, use CCS_BIND_APP= instead.</p>	<p>the running container instance. To list multiple Cloud services, include them as part of the same environment variable.</p>
<p>(Deprecated) CCS_SSH_KEY= <i><public_ssh_key></i></p>	<div>This environment variable has been deprecated</div> <p>Use cf ic exec or cf ic attach for external access to your containers instead. For more information, see Logging in to a container with exec.</p> <p>To add an SSH key to a container when you create it, you can use CCS_API_KEY.</p>
<p>LOG_LOCATIONS= <i><path_to_file></i></p>	<p>To add a log file to be monitored in the container, include the LOG_LOCATIONS environment variable with a path to the log file.</p>

Example:

```
cf ic group create -e CCS_BIND_SRV=<service_instance_name> -
```

--env-file *ENVFILE*

(Optional) Import environment variables from a file where *ENVFILE* is the path to your file on your local directory. Every line in the file represents one key=value pair.

Example

```
KEY1=VALUE1
KEY2=VALUE2
```

--http_monitor_enabled

(Optional) The default value is `true`. When automatic recovery is enabled for a container group and the group has been running for 10 minutes, the group's load balancer starts to regularly check the health of each container instance in the group via HTTP requests. If a container instance does not respond within 100 seconds, it is marked as inactive. Inactive container instances are removed from the group and re-created by auto-recovery. Auto-recovery attempts to recover container instances in a group 3 times. After the third attempt, auto-recovery does not recover any container instances in the group for 60 minutes. After 60 minutes, the auto-recovery process starts again. Some apps might not require an HTTP health check, such as an app that uses TCP for communication instead of HTTP. To disable the HTTP monitor, set the option to `false`. Example:

```
--http_monitor_enabled=false
```

--http_monitor_path

(Optional) The default value is `/`. When `--http_monitor_enabled` is enabled, you can use this option to set a custom path for the health check, such as `--http_monitor_path="/health/path"`.

--http_monitor_rc_list

(Optional) When `--http_monitor_enabled` is enabled, you can specify the list of response codes for the health check. The default value includes the 200, 201, 202, 204, 300, 301, 302, 401, 403, and 404 codes. Any response not in the list of valid responses changes the container instance's state to `INACTIVE` by the load balancer. An inactive container does not receive any load and the state of the container group changes to `HAS_INACTIVE_INSTANCES`. If none of the containers are `ACTIVE`, then the state of the container group changes to `NO_ACTIVE_INSTANCES`. To learn more about states, see [Container states in IBM Cloud Container](#)

[Service](#).

IMAGE

(Required) The image to include in each container instance in the container group. You can list commands after the image, but do not put any options after the image; include all options before you specify an image.

If you are using an image in your organization's private Cloud images registry, specify the image in the following format:

`registry.DomainName/NAMESPACE/IMAGE`

If you are using an image that is provided by IBM Cloud Container Service, do not include your organization's namespace. Specify the image in the following format: `registry.DomainName/IMAGE`

--ip IP_ADDRESS

(Optional) If you have an available IP address, bind a public IP address to your container group. You must also expose the port with the `-p` flag. A container group can have either a public route or a public IP address but not both. To unbind the IP address and return the IP to the space's quota, remove the container group and recreate the group without the IP address. To request an IP address prior to creating a container group, run `cf ic ip request`.

Note

Binding a public IP address does not include a SSL certificate which encrypts the data that are sent to and from your container group. If your app requires SSL encryption, you can either implement your own SSL solution or map a public route to your container group instead of binding a public IP address. Public routes already include a SSL certificate, so you can access your container group with HTTPS without any additional configuration.

-m MEMORY, --memory MEMORY

(Optional) Enter a memory limit for your container in MB. The memory limit is part of the container size that defines the maximum amount of memory

and disk space a container gets on the compute host during runtime. After a container size is assigned, the value cannot be changed. Available sizes in IBM Cloud Container Service are:

Pico (64 MB memory, 4 GB disk space)

Nano (128 MB memory, 8 GB disk space)

Micro (256 MB memory, 16 GB disk space)

Tiny (512 MB memory, 32 GB disk space)

Small (1024 MB memory, 64 GB disk space)

Medium (2048 MB memory, 128 GB disk space)

Large (4096 MB memory, 256 GB disk space)

X-Large (8192 MB memory, 512 GB disk space)

2X-Large (16384 MB memory, 1 TB disk space)

If you do not set a size for your container, each container instance is created with 256 MB.

Important

Enter the container memory in MB without the unit label. For example, if you want to create a pico container, enter `-m 64`.

--max *MAX*

(Optional) The maximum number of instances. The default is 2.

--min *MIN*

(Optional) The minimum number of instances. The default is 1.

-n *HOST*, --hostname *HOST*

(Optional) The host name, such as *mycontainerhost*. Do not include underscores (_) in the host name. The host and the domain combined form the full public route URL, such as `http://mycontainerhost.AppDomainName`. When you review the details of a container group with the `cf ic group inspect` command, the host and the domain are listed together as the route.

--name *NAME*

(Required) Assign a name to the group. -n is deprecated.

Tip

The container name must start with a letter, and then can include uppercase letters, lowercase letters, numbers, periods (.), underscores (_), or hyphens (-).

-p *PORT*, --publish *PORT*

(Required) Expose the port for HTTP and private network traffic. If you do not expose a port, the container group runs, but the container group cannot be accessed through the group load balancer on either the public or private network.

For container groups, you cannot include multiple ports. If multiple ports are listed, only the last listed port is exposed. The other ports are ignored by the command.

When you bind a route, containers in your group must listen for HTTP traffic on the group's exposed port. Non-HTTP ports cannot be exposed publicly. When HTTPS traffic arrives on the exposed port, the (Go)Router completes the HTTPS termination. Then, the HTTP protocol is used on the private virtual network between the (Go)Router and the containers. When you bind an IP address, containers in your group can listen for HTTP or HTTPS traffic, but you can still expose only one port.

If a port is specified in the Dockerfile for the image that you are using, include that port.

Tip

For the IBM certified Liberty Server image or a modified version of this image, enter the port 9080.

For the IBM certified Node.js image or a modified version of this

image, enter any port, such as port 8000.

--session_affinity

(Optional) The default value is `false`. When `--http_monitor_enabled` is set to `true`, you can set the `--session_affinity` option to `true` to allow IBM Cloud Container Service to create and manage a session cookie on behalf of the app. This option is not available if `--http_monitor_enabled` is set to `false` or if you are using TCP for communication instead of HTTP.

--volume VOLUME:/DIRECTORY_PATH[:ro]

(Optional) Attach a volume to a container by specifying the details in the following format `VOLUME:/DIRECTORY_PATH[:ro]`.

VOLUME: The volume ID or name.

DIRECTORY_PATH: The absolute path to the directory in the container.

ro: Optional. Specifying `ro` makes the volume read-only instead of the default read/write.

Example

The following example is a request to create a scaling group with a public route.

```
cf ic group create --auto -d AppDomainName -n mycontainerhost --nam
```

The following example is a request to create a scaling group with a public IP address.

```
cf ic group create --auto --ip 198.51.100.27 --name my_container_gr
```

cf ic group inspect GROUP

Purpose

See the detailed information that was specified for a container group when it was created, such as the environment variables, ports, or memory.

Parameters

GROUP

(Required) The group ID or name.

Example

The following example is a request to inspect the group *my_container_group*.

```
cf ic group inspect my_container_group
```

cf ic group instances GROUP

Purpose

List all containers with their ID, name, IP address and public port that are part of a container group.

Parameters

GROUP

(Required) The group ID or name.

Example

The following example is a request to retrieve all containers of the group *my_container_group*.

```
cf ic group instances my_container_group
```

cf ic group list [-q]

Purpose

List the container groups in a space.

Parameters

-q

(Optional) List the container group ID of each container group only.

Example

```
bx ic groups [-q]
```

cf ic group rm [-f] GROUP [GROUP]

Purpose

Remove one or more container groups from a space.

Parameters

-f, --force

(Optional) Forces the removal of a running or failed container.

GROUP

(Required) The group ID or name of at least one container group.

Example

The following example is a request to remove a container group, where *my_container_group* is the name of the group.

```
cf ic group rm my_container_group
```

cf ic group update [--desired DESIRED] [-e "KEY=VALUE"] [--max MAX] [--min MIN] GROUP

Purpose

Use the `cf ic group update` command to perform the following updates on your group.

Update the desired number of instances that run in your container group.

Add new environment variables to the group.

Update the value of an existing environment variable.

Note

You can perform one update at a time only.

Tip

To update the host name or domain for a container group, use `cf ic route map [-d DOMAIN] [-n HOST] GROUP`.

Parameters

--desired *DESIRED*

(Optional) The number of instances that you require. The default is 2.

-e "KEY=VALUE"

(Optional) Add a new environment variable to the container group or change the value of an existing environment variable.

--max *MAX*

(Optional) The maximum number of instances. The default is 2.

--min *MIN*

(Optional) The minimum number of instances. The default is 1.

Example

The following example is a request to update the desired number of container instances in a container group named *my_container_group*.

```
cf ic group update --desired 5 my_container_group
```

The following example is a request to add a new environment variable *environment* with the value *Linux* to a container group that is named *my_container_group*.

```
cf ic group update -e "environment=Linux" my_container_group
```

cf ic images [-a] [-f **CONDITION**] [--no-trunc] [q]

Purpose

See a list of all the available images in the organization's private Cloud images registry. For more information, see the [images](#) command in the Docker help.

Parameters

-a, --all

(Optional) Includes all of the image layers for each image in your organization's registry, not just the most recent layer.

-f, --filter *CONDITION*

(Optional) Filter the list of images by the condition provided.

--no-trunc

(Optional) Do not truncate the output.

-q, --quiet

(Optional) Display the numeric IDs only.

Responses

You receive a list of images that includes the image ID, the created date, and the image name.

Example

The following example is a request to receive a list of available images for the organization.

```
cf ic images
```

cf ic info [--json]

Purpose

Receive a set of information that describes the state of the container cloud service instance.

Parameters

--json

(Optional) Structure the output of the command in JSON format.

Responses

You receive a readout that includes information about the containers limit, containers usage, containers that are running, memory limit, memory usage, floating IP limit, floating IP usage, CCS host URL, registry host URL, and debug mode status.

Example

The following example is a request to receive information about the container cloud service instance.

```
cf ic info
```

cf ic init

Purpose

Refresh the bearer token by running this command to log in to Cloud Foundry again and reinitialize the CLI. Running `cf ic init` has the same effect as running `cf ic`

login again.

Example

```
cf ic init
```

cf ic inspect [IMAGE | images | CONTAINER]

Purpose

Retrieve information about one or more containers or container images. Only one of IMAGE, images, or CONTAINER options can be specified at a time. For more information, see the [inspect](#) command in the Docker help.

Parameters

CONTAINER

(Required) Get detailed information about a specific container by specifying the container name or ID. You can retrieve information for multiple containers by listing their names or IDs in the command with a space in between.

IMAGE

(Required) Get detailed information about a specific image by specifying the full private Cloud registry path to your image. You can retrieve information for multiple images by listing each registry path in the command with a space in between

images

(Required) Get detailed information about all of the images in your registry.

Example

The following example is a request to inspect a container that is named `my_container`.

```
cf ic inspect my_container
```

cf ic ip bind ADDRESS CONTAINER

Purpose

Bind an available floating IP address to a single container.

Note

To bind a floating IP address to a container group, use the `--ip` flag in the `cf ic group create` command.

Parameters

ADDRESS

(Required) The IP address to bind to the single container.

CONTAINER

(Required) The container ID or name to bind to the IP address.

Example

The following example is a request to bind the IP address 192.0.2.23 to the container with name *my_container*.

```
cf ic ip bind 192.0.2.23 my_container
```

cf ic ip list [-q]

Purpose

List the available floating IP addresses for the organization and the IDs for the single containers or groups that are bound to those IP addresses.

Parameters

-q

(Optional) List only the IP addresses, without the IDs for the single containers or groups that are bound to those IP addresses.

Responses

You receive a list of IP addresses and the ID of the single container or container group they are linked to. If the IP address is unused, no container ID is shown.

Example

The following example is a request to receive a list of all IP addresses for the organization, whether they are available to use or not.

```
cf ic ip list -q
```

cf ic ip release ADDRESS

Purpose

Release a floating IP address. After a floating IP address is released, it cannot be bound to a single container or container group in the space.

Parameters

ADDRESS

(Required) The IP address to release.

Example

The following example is a request to release the IP address 192.0.2.23

```
cf ic ip release 192.0.2.23
```

cf ic ip request [-q]

Purpose

Request a new floating IP.

Parameters

-q

(Optional) Only the IP address is displayed in the command output. No message text is included.

Example

The following example is a request for a new floating IP address.

```
cf ic ip request
```

cf ic ip unbind ADDRESS CONTAINER

Purpose

Unbind a floating IP address from a single container.

Note

To unbind the IP address from a container group, remove the container group and recreate the group without the IP address.

Parameters

ADDRESS

(Required) The IP address that is being unbound.

CONTAINER

(Required) The container ID or name that is being unbound.

Example

The following example is a request to unbind the IP address 192.0.2.23 from the container with name *my_container*.

```
cf ic ip unbind 192.0.2.23 my_container
```

cf ic kill [-s CMD] CONTAINER [CONTAINER]

Purpose

Stop a running process in one or more containers without stopping the container. For more information, see the [kill](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container ID or name of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

-s, --signal CMD

(Optional) Send any command to the process that is running in the container.

Example

The following example is a request to kill the process in a container that is named *my_container*.

```
cf ic kill my_container
```


cf ic login

Note

Before logging in, you must ensure that the Cloud Foundry command tool (cf) is installed, as well as the Docker CLI (docker), and that they are both configured in your environment path.

Purpose

Log in to the CLI to use the full capabilities of IBM Cloud Container Service.

Tip

To switch to another region, specify the API endpoint for that region when you log in to the Cloud Foundry CLI, not the IBM Cloud Container Service CLI.

US South:

```
cf login -a api.ng.bluemix.net
```

United Kingdom:

```
cf login -a api.eu-gb.bluemix.net
```

Example

The following example is a standard login.

```
cf ic login
```

cf ic logs [-f | --since TIMESTAMP | --tail STRING | -t] CONTAINER

Purpose

Show the output or error logs for a running container. For more information, see the

[logs](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID.

-f, --follow

(Optional) Include new output from the STDOUT and STDERR log files for the container.

--since TIMESTAMP

(Optional) Show all logs since the timestamp.

--tail STRING

(Optional) Specify the number of the most recent lines of logs to show. The default is to show all logs.

-t, --timestamps

(Optional) Display timestamps on the logs.

Example

The following example is a request to get the output log for a container that is named *my_container*.

```
cf ic logs -o my_container
```

cf ic namespace [get | set NAME]

Purpose

Set or retrieve the name of the private Cloud image registry for the organization that the user has logged in with.

Parameters

get

(Optional) Request the namespace for your organization's private images registry.

set NAME

(Optional) A one-time only function to set the namespace for your organization's private image registry, if it is not already set. After you set the namespace, log out and in to the IBM Cloud Container Service CLI before

you continue.

Restriction

You cannot use a hyphen (-) in the name of your registry namespace.

Example

The following example is a request to query the namespace for the organization that the user logged in to.

```
cf ic namespace get
```

cf ic pause **CONTAINER [CONTAINER]**

Purpose

Pause all processes within one or more running containers. For more information, see the [pause](#) command in the Docker help. To start the container again, run the [unpause](#) command.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

Example

The following example is a request to pause a container that is named *my_container*.

```
cf ic pause my_container
```

cf ic port **CONTAINER [PRIVATE_PORT[/PROTOCOL]]**

Purpose

List all port mappings for a container or retrieve the public port and IP address that is mapped to your private port. For more information, see the [port](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container ID or name.

PRIVATE PORT

(Optional) The private port for which you want to look up the assigned public port and IP address.

PROTOCOL

(Optional) The protocol that was assigned to your private port. It can be either *tcp* or *udp*.

Example

The following example is a request to list all port and IP mappings for a container that is named *my_container*.

```
cf ic port my_container
```

System Output

```
9080/tcp -> 192.0.2.25:9080
```

In this example, 9080/tcp is your private port that is using the TCP/IP protocol. It is mapped to the public IP address 192.0.2.25 and public port 9080.

cf ic ps [-a] [--filter env=*SEARCH_CRITERIA*] [-l *NUM*] [-q] [-s]

Purpose

Retrieve a list of containers that are running in the space. By default, only running containers are displayed. For more information, see the [ps](#) command in the Docker help.

Parameters

-a, --all

(Optional) Show all containers in the space, both running and stopped.

--filter env=*SEARCH_CRITERIA*

(Optional) Search containers that have a specific environment variable value. You can filter your containers by any environment variable key or value that is listed in the *Env* section of your CLI response when you inspect

a container. Replace *SEARCH_CRITERIA* with the key or value you are looking for. Your search criteria does not need to be an exact match.

-l NUM, --limit NUM

(Optional) List the most recently created containers, where *NUM* is the number of the most recently created containers that you want to return.

For example, if you created containers node1 through to node5 sequentially, the option `--limit 2` returns node4 and node5 because they are the last two containers that were created.

-q, --quiet

(Optional) Display only container IDs.

-s, --size

(Optional) List the sizes of the containers.

Examples

The following example is a request to see all running and stopped containers.

```
cf ic ps -a
```

cf ic rename OLD_NAME NEW_NAME

Purpose

Rename a container. For more information, see the [rename](#) command in the Docker help.

Parameters

NEW_NAME

(Required) A new name for the container.

OLD_NAME

(Required) The old name of the container.

Example

The following example is a request to rename a container that is currently named *my_container* to *container_renamed*.

```
cf ic rename my_container container_renamed
```

cf ic reprovision [-f] [AVAILABILITY_ZONE]

Purpose

Change the IBM Cloud Container Service availability zone for a space. The original quota for the space is maintained. By deploying your app in multiple container groups in different availability zones, you are creating higher availability for your app in case of hardware failures.

Attention

When you run this command, all of your single containers and groups, public IP addresses and volumes in this space will not be migrated to the new availability zone and will be removed during the migration process. Images will not be affected. [Backup your volumes](#) to ensure you can restore data after you migrated to the new availability zone.

Parameters

AVAILABILITY_ZONE

(Optional) The name of the IBM Cloud Container Service availability zone where your containers are deployed. If no IBM Cloud Container Service availability zone is specified, the default IBM Cloud Container Service availability zone that is set for the region is used.

Important

If an IBM Cloud Container Service availability zone is included, the `-f` option must also be included.

Table 23. IBM Cloud Container Service availability zones

Region	Data center location	IBM Cloud Container Service availability zone names
US South	Dallas 1	dal09-01

Region	Data center location	IBM Cloud Container Service availability zone names
US South	Dallas 2	dal09-02
US South	Dallas 3	dal10-03
United Kingdom	London	lon02-01
United Kingdom	Amsterdam	ams03-01

-f, --force

(Optional) Forces the re-creation of IBM Cloud Container Service in the Cloud space.

Example

The following example is a request to change the IBM Cloud Container Service availability zone for the current space to dal09-02.

```
cf ic reprovision -f dal09-02
```

cf ic restart CONTAINER [CONTAINER] [-t SECS]

Purpose

Restart one or more containers. For more information, see the [restart](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

-t SECS, --time SECS

The number of seconds to wait before the container is restarted.

Example

The following example is a request to restart a container that is named *my_container*.

```
cf ic restart my_container
```

cf ic rm [-f] CONTAINER [CONTAINER]

Purpose

Remove one or more containers. For more information, see the [rm](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

-f, --force

(Optional) Forces the removal of a running or failed container.

Example

The following example is a request to remove a container that is named *my_container*.

```
cf ic rm my_container
```

cf ic rmi [-R REGISTRY] IMAGE [IMAGE]

Purpose

Remove one or more images from the logged-in organization's private Cloud images registry. For more information, see the [rmi](#) command in the Docker help.

Parameters

IMAGE

(Required) The full private Cloud registry path to the image that you want to remove. If a tag is not specified in the image path, the image tagged `latest` is deleted by default. You can run this command with multiple images by listing each private Cloud registry path in the command with a space in between.

-R REGISTRY, --registry REGISTRY

(Optional) Change the registry host. The default is to use the registry that

you used with the **login** command.

Example

The following example is a request to remove the image `mynamespace/myimage:latest`:

```
cf ic rmi registry.DomainName/mynamespace/myimage:latest
```

cf ic route map [-d DOMAIN] [-n HOST] GROUP

Purpose

Establish the route for internet traffic to use to access a container group. You can use this command to establish a new route or update an existing route.

Parameters

-d DOMAIN, --domain DOMAIN

(Optional) The domain name for the route, which is the second part of the full public route URL. In most cases, the domain is `AppDomainName`. You can also use this parameter to specify a custom domain. If you use a custom domain, do not include underscores (`_`) in the domain name.

GROUP

(Required) The group ID or name.

-n HOST, --hostname HOST

(Optional) The host name for the route. Do not include underscores (`_`) in the host name. The host name is the first part of the full public route URL, such as `mycontainerhost` in the URL `mycontainerhost.AppDomainName`.

Example

The following example is a request to map the route for a container group that is named `my_container_group`. The host name is `my_host` and `AppDomainName` is the domain.

```
cf ic route map -d AppDomainName -n my_host my_container_group
```

cf ic route unmap [-d DOMAIN] [-n HOST] GROUP

Purpose

Unmap the route from a container group. A group that does not have a route is not

accessible from the Internet.

Parameters

-d DOMAIN, --domain DOMAIN

(Optional) The domain name for the route.

GROUP

(Required) The group ID or name.

-n HOST, --hostname HOST

(Optional) The host name for the route.

Example

The following example is a request to unmap the route for the group that is called `my_container_group`, where `my_host` is the host name and `AppDomainName` is the domain.

```
cf ic route unmap -d AppDomainName -n my_host my_container_group
```

cf ic run [-p PORT] [-P] [-d] [-e ENV] [--env-file ENVFILE] [-it] [--link NAME:ALIAS] [-m MEMORY] --name NAME [--volume VOLUME:/DIRECTORY_PATH] IMAGE [CMD [CMD ...]]

Purpose

Start a new container in the container cloud service from an image name. By default, container in Cloud run in detached mode. For more information, see the [run](#) command in the Docker help.

Note

You must ensure that the Cloud Foundry command tool is installed and that you have a Cloud Foundry token. Successful login by using `cf login` and `cf ic login` generates the required token.

Parameters

CMD

(Optional) The command and arguments are passed the container to

execute. This command must be a long-running command. Do not use a short-lived command, for example, **/bin/date**, because it might cause the container to crash.

Example long-running commands, assuming that you are using the `ibmnode` image that is provided in the IBM Cloud Container Service registry:

```
cf ic run --name my_container registry.DomainName/ibmnode pi
```

```
cf ic run --name my_container registry.DomainName/ibmnode --
```

```
cf ic run --name my_container registry.DomainName/ibmnode --
```

Note

The CMD and its arguments must come at the end of the `cf ic group create` command line.

-d

(Optional) Run the container in detached mode. Containers run in detached mode by default when you use `cf ic run`. In detached mode, when the root process in the container exits, the container also exits. For example, when you deploy a container from the `ibmnode` image, your container shuts down immediately after it is built as no long running process exists.

Tip

If you choose to use `docker run` instead of `cf ic run`, the `-d` option is required.

-e ENV, --env ENV

(Optional) Set a number of environment variables where ENV is a key=value pair. List multiple keys separately and if you include quotation marks, include them around both the environment variable name and the

value.

Example

```
-e "key1=value1" -e "key2=value2" -e "key3=value3"
```

Table 24. Suggested environment variables

Environment variable	Description
CCS_BIND_APP= <appname>	Some Cloud services do not support direct binding to a container. In this case, you need to create a Cloud Foundry app and bind the Cloud service to it. Then, you bind the app to your container by using CCS_BIND_APP. The Cloud Foundry app acts as a bridge and allows Cloud to inject your bridge app's VCAP_SERVICES information into the running container instance.
CCS_BIND_SRV= <service_instance_name1>, <service_instance_name2>	To bind a Cloud service directly to a container without using a bridge app, use CCS_BIND_SRV. This binding allows Cloud to inject the VCAP_SERVICES information into the running container instance. To list multiple Cloud services, include them as part of the same environment variable.
(Deprecated) CCS_SSH_KEY= <public_ssh_key>	This environment variable

Environment variable	Description
	<p>has been deprecated</p> <p>Use <code>bx ic exec</code> or <code>bx ic attach</code> for external access to your containers instead. For more information, see Logging in to a container with exec.</p> <p>To add an SSH key to a container when you create it, you can use <code>CCS_API_KEY</code>.</p>
<code>LOG_LOCATIONS=<path_to_file></code>	To add a log file to be monitored in the container, include the <code>LOG_LOCATIONS</code> environment variable with a path to the log file.

--env-file *ENVFILE*

(Optional) Import environment variables from a file where *ENVFILE* is the path to your file on your local directory. Every line in the file represents one key=value pair.

Example

```
KEY1=VALUE1
KEY2=VALUE2
```

IMAGE

(Required) The image to include in the container. You can list commands after the image, but do not put any options after the image; include all options before you specify an image.

If you are using an image in your organization's private Cloud registry, specify the image in the following format:

`registry.DomainName/NAMESPACE/IMAGE`

If you are using an image that is provided by IBM Cloud Container Service, specify the image in the following format: `registry.DomainName/IMAGE`

-it

(Optional) Run the container in interactive mode. After the container is created, keep the standard input display. Type `exit` to exit.

--link NAME:ALIAS

(Optional) Whenever you want a container to communicate with another container that is running, you can address it using an alias for the host name. For more information, see [Linking single containers with the command line interface \(CLI\)](#).

-m MEMORY, --memory MEMORY

(Optional) Enter a memory limit for your container in MB. The memory limit is part of the container size that defines the maximum amount of memory and disk space a container gets on the compute host during runtime. After a container size is assigned, the value cannot be changed. Available sizes in IBM Cloud Container Service are:

Pico (64 MB memory, 4 GB disk space)

Nano (128 MB memory, 8 GB disk space)

Micro (256 MB memory, 16 GB disk space)

Tiny (512 MB memory, 32 GB disk space)

Small (1024 MB memory, 64 GB disk space)

Medium (2048 MB memory, 128 GB disk space)

Large (4096 MB memory, 256 GB disk space)

X-Large (8192 MB memory, 512 GB disk space)

2X-Large (16384 MB memory, 1 TB disk space)

If you do not set a size for your container, each container instance is created with 256 MB.

Important

Enter the container memory in MB without the unit label. For example, if you want to create a pico container, enter `-m 64`.

--name *NAME*

(Required) Assign a name to the container.

Tip

The container name must start with a letter, and then can include uppercase letters, lowercase letters, numbers, periods (.), underscores (_), or hyphens (-).

-p *PORT*, --publish *PORT*

(Optional) If you want to make your app accessible from the Internet, you must expose a public port. When you expose a public port, you create a Public Network Security Group for your container that allows you to send and receive public data on the exposed port only. All other public ports are closed and cannot be used to access your app from the internet. You can include multiple ports with multiple `-p` options. Ports cannot be mapped or forwarded. If you do not expose a port, your container is accessible from the private container network only. You can use the assigned private IP address to communicate with your container on the private network.

If a port is specified in the Dockerfile for the image that you are using, include that port.

Tip

For the IBM certified Liberty Server image or a modified version of this image, enter the port 9080.

For the IBM certified Node.js image or a modified version of this image, enter any port, such as port 8000.

You can choose between UDP and TCP to indicate the IP protocol that you want to use. If you do not specify a protocol, your port is automatically exposed for TCP traffic.

Examples

```
-p 9080  
-p 9080/udp  
-p 9080/udp -o 8000/tcp
```

If you have an existing IP address in the space that you want to bind to the container, you can specify the IP address with the port information. Include the IP address, the port, or the IP address and the container port number, such as `<ip-address>:<port_on_IP>:<container-port>`. The port on the IP address must match the container port exposed in the Dockerfile.

Example

```
-p 192.0.2.23:9080:9080
```

Note

Binding a public IP address does not include a SSL certificate which encrypts the data that are sent to and from your single container. If your app requires SSL encryption, you can either implement your own SSL solution or use a container group instead of a single container. Container groups are bound to a public route that already includes a

SSL certificate, so you can access your container group with HTTPS without any additional configuration. For more information, see [Creating a container group with the GUI](#).

-P

(Optional) Automatically expose the ports that are specified in the image's Dockerfile for HTTP traffic.

--volume VOLUME:/DIRECTORY_PATH[:ro]

(Optional) Attach a volume to a container by specifying the details in the following format `VOLUME: /DIRECTORY_PATH[:ro]`.

VOLUME The volume ID or name.

DIRECTORY_PATH: The absolute path to the directories in the container.

ro Optional. Specifying *ro* makes the volume read-only instead of the default read/write.

Examples

The following example is a valid request to create and start a new container by using the `my_namespace/nginx` image, with the name `my_container` and a 1024 MB memory limit, where `my_namespace` is the logged in users associated namespace.

```
cf ic run -m 1024 --name my_container registry.DomainName/my_namespa
```

The following example is a valid request to create and start a new container by using the `my_namespace/blog` image, passing in some credentials as environment variables, where `my_namespace` is the logged in users associated namespace.

```
cf ic run -e USER=johnsmith -e PASS=password --name my_container reg
```

The following example is a valid request to add a volume to a container by using the `my_namespace/blog` image, where `my_namespace` is the logged in users associated namespace.

```
cf ic run --name my_container --volume VolId1:/first/path --volume V
```

cf ic start CONTAINER [CONTAINER]

Purpose

Start one or more stopped containers. For more information, see the [start](#) command in the Docker help. To stop a container, see the [stop](#) command.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

Example

The following example is a request to start a container that is named *my_container*.

```
cf ic start my_container
```

cf ic stats --no-stream CONTAINER [CONTAINER]

Purpose

For one or more containers, view live usage statistics for the container resources. Data is displayed as it is generated in real time. Use CTRL+C to exit. For more information, see the [stats](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID.

--no-stream

(Required) Display the latest results. No new information that is generated after running the command is displayed.

Example

The following example is a request for the most recent statistics about a container.

```
cf ic stats --no-stream my_container
```

cf ic stop CONTAINER [CONTAINER] [-t SECS]

Purpose

Stop one or more running containers. For more information, see the [stop](#) command in

the Docker help. To start a container, see the [start](#) command.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names and IDs in the command with a space in between.

-t SECS, --time SECS

The number of seconds to wait before the container is killed.

Example

The following example is a request to stop a container that is named *my_container*.

```
cf ic stop my_container
```

cf ic top CONTAINER [CONTAINER] [ps options]

Purpose

List the processes that are running in the container.

Other than the native `top` command in Linux, the `cf ic top` command displays the output of the Linux `ps` command for the main processes that are running inside a container. As a consequence, the memory and CPU usage is not updated in your CLI output when it changes for a process.

Note

If you need to see the output of the native Linux `top` command, [exec](#) in to your container and run the `top` command.

For more information, see the [top](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID.

ps options

(Optional) Use the options that are provided by the native Linux `ps` command to further detail or filter the output of the `cf ic top` command.

Example

The following example is a request to display detailed information for all current processes that are running inside a container named *my_container*.

```
cf ic top my_container -aux
```

cf ic unbind-service GROUP SERVICE_INSTANCE

Purpose

Remove a service from a running container group. This command is only available to container groups. Single containers must remove the container and create a new container without the service. To bind a service, see the [bind-service](#) command.

Parameters

GROUP

(Required) The group ID or name.

SERVICE_INSTANCE

(Required) The name of the service instance to be removed from the container group.

Example

The following example is a request to unbind an instance of the Data Cache service named *MyDataCache* from the container *my_container_group*.

```
cf ic unbind-service my_container_group MyDataCache
```

cf ic unpause CONTAINER [CONTAINER]

Purpose

Unpause all processes within one or more running containers. For more information, see the [unpause](#) command in the Docker help. To pause a container, see the [pause](#) command.

Parameters

CONTAINER

(Required) The container name or ID of at least one container. You can run this command with multiple containers by listing their names or IDs in the command with a space in between.

Example

The following example is a request to unpause a container that is named *my_container*.

```
cf ic unpause my_container
```

cf ic unprovision [-f]

Purpose

Delete IBM Cloud Container Service from the Cloud space that you are logged into.

Attention

When you run this command, all your single containers and container groups are lost. Your space is still available in Cloud. To start using IBM Cloud Container Service again, you must run [cf ic login](#) to provision the IBM Cloud Container Service again.

Parameters

-f, --force

(Optional) Forces the deletion of IBM Cloud Container Service from the Cloud space.

Example

The following example is a request to delete IBM Cloud Container Service from the space.

```
cf ic unprovision
```

cf ic update

Purpose

Check to see if there is a more recent version of the IBM Cloud Container Service plug-in than the version you have installed.

Example

```
bx ic update
```

cf ic version

Purpose

Shows the version of Docker and the IBM Cloud Container Service API. To see the version of the installed Cloud Foundry plug-in for IBM Cloud Container Service, run `cf plugins` or `cf ic info`. For more information, see the [version](#) command in the Docker help.

Example

```
bx ic version
```

cf ic volume create VOLNAME [FSNAME]

Purpose

Create the volume.

Parameters

FSNAME

(Optional) The file share name. If no file share is available or named the volume will be built on the space's default file share.

VOLNAME

(Required) The volume name. The name can contain uppercase letters, lowercase letters, numbers, underscores (_), and hyphens (-).

Example

The following example is a request to create a volume.

```
cf ic volume create my_volume my_fileshare
```

cf ic volume fs-create FSNAME FSSIZE FSIOPS

Purpose

Create a file share.

Parameters

FSIOPS

(Required) The file share IOPS. Valid values are 0.25, 2 or 4 IOPS per GB.

FSNAME

(Required) The file share name. The name can contain uppercase letters, lowercase letters, numbers, underscores (_), and hyphens (-).

FSSIZE

(Required) The file share system size in GB. Use `cf ic volume fs-flavor-list` to list all valid sizes

Example

```
cf ic volume fs-create my_fileshare 20 0.25
```

cf ic volume fs-flavor-list

Purpose

Lists the file share sizes that are available to use.

Example

```
bx ic volume-fs-flavors
```

cf ic volume fs-inspect FSNAME

Purpose

Inspect the file share.

Parameters

FSNAME

(Required) The file share name.

Example

```
cf ic volume fs-inspect my_fileshare
```

cf ic volume fs-list

Purpose

List the file shares in your organization.

Example

```
bx ic volume-fs
```

cf ic volume fs-rm FSNAME

Purpose

Remove a file share.

Parameters

FSNAME

(Required) The file share name.

Example

```
cf ic volume fs-rm my_fileshare
```

cf ic volume inspect VOLNAME

Purpose

Inspect the volume.

Parameters

VOLNAME

(Required) The volume name.

Example

The following example is a request to inspect the volume, where *my_volume* is the name of the volume.

```
cf ic volume inspect my_volume
```

cf ic volume list [--table]

Purpose

List the volumes.

Parameters

--table

(Optional) Structures the list output in a table format.

Example

The following example is a request to list all the volumes.

```
bx ic volumes [--table]
```

cf ic volume rm VOLNAME

Purpose

Remove the volume.

Parameters

VOLNAME

(Required) The volume name.

Example

The following example is a request to remove the volume, where *my_volume* is the name of the volume.

```
cf ic volume rm my_volume
```

cf ic wait CONTAINER [CONTAINER]

Purpose

Wait for a container to exit. During this waiting time your command line does not return and you cannot enter commands. As soon as the container exits or stops, the exit code is printed in the CLI output. For more information, see the [wait](#) command in the Docker help.

Parameters

CONTAINER

(Required) The container name or ID.

Example

The following example is a request to wait for a container that is called *my_container* to exit. As soon as it exits, the exit code is printed in the CLI.

```
cf ic wait my_container
```

cf ic wait-status CONTAINER

Purpose

Wait for a single container or container group to reach a non-transient state. During this waiting time your command line does not return and you cannot enter commands. As soon as the container reaches a non-transient state, an **OK** message is displayed. For single containers, the non-transient states include Running, Shutdown, Crashed, Paused, or Suspended. For container groups, the non-transient states include CREATE_COMPLETE, UPDATE_COMPLETE, or FAILED.

Parameters

CONTAINER

(Required) The container or group name or ID.

Example

The following example is a request to wait for a container that is called *my_container* until it reaches a non-transient state.

```
cf ic wait-status my_container
```

Migrating cf ic commands to bx ic commands

When you change the deprecated cf ic commands to bx ic commands, note that some multi-word commands are different. For example, some multi-word commands are flipped and some include hyphens (-).

cf ic command	bx ic command
cf ic attach	bx ic attach
cf ic bind-service	bx ic service-bind
cf ic build	bx ic build

cf ic command	bx ic command
cf ic cp	bx ic cp
cf ic cpi	bx ic cpi
cf ic exec	bx ic exec
cf ic group create	bx ic group-create
cf ic group inspect	bx ic group-inspect
cf ic group instances	bx ic group-instances
cf ic group list	bx ic groups
cf ic group rm	bx ic group-remove
cf ic group update	bx ic group-update
cf ic images	bx ic images
cf ic info	bx ic info
cf ic init	bx ic init
cf ic inspect	bx ic inspect
cf ic ip bind	bx ic ip-bind
cf ic ip list	bx ic ips
cf ic ip release	bx ic ip-release
cf ic ip request	bx ic ip-request
cf ic ip unbind	bx ic ip-unbind
cf ic kill	bx ic kill
cf ic login	bx ic init
cf ic logs	bx ic logs
cf ic namespace get	bx ic namespace-get
cf ic namespace set	bx ic namespace-set
cf ic pause	bx ic pause

cf ic command	bx ic command
cf ic port	bx ic port
cf ic ps	bx ic ps
cf ic rename	bx ic rename
cf ic reprovision	bx ic reprovision
cf ic restart	bx ic restart
cf ic rm	bx ic rm
cf ic rmi	bx ic rmi
cf ic route map	bx ic route-map
cf ic route unmap	bx ic route-unmap
cf ic run	bx ic run
cf ic start	bx ic start
cf ic stats	bx ic stats
cf ic stop	bx ic stop
cf ic top	bx ic top
cf ic unbind-service	bx ic service-unbind
cf ic unpause	bx ic unpause
cf ic unprovision	bx ic unprovision
cf ic update	bx ic update
cf ic version	bx ic version
cf ic volume create	bx ic volume-create
cf ic volume fs-create	bx ic volume-fs-create
cf ic volume fs-flavor-list	bx ic volume-fs-flavors
cf ic volume fs-inspect	bx ic volume-fs-inspect
cf ic volume fs-list	bx ic volume-fs

cf ic command	bx ic command
cf ic volume fs-rm	bx ic volume-fs-remove
cf ic volume inspect	bx ic volume-inspect
cf ic volume list	bx ic volumes
cf ic volume rm	bx ic volume-remove
cf ic wait	bx ic wait
cf ic wait-status	bx ic wait-status

Troubleshooting single and scalable containers (Deprecated)

If you have problems or questions when you are using IBM® Cloud Container Service, you can get help by searching for information or by asking questions through a forum. You can also open a support ticket.

Attention

Single and scalable containers are deprecated. Support will continue in IBM Cloud Public until 5 December 2017 and in IBM Cloud Local and Dedicated until 20 June 2018. Start using Kubernetes clusters today to deploy secure, highly available apps. [Learn more about Kubernetes and how to migrate your apps.](#)

In this page:

- [Debugging containers and reviewing logs](#)
- [Troubleshooting the CLI setup](#)
- [Troubleshooting quota](#)
- [Troubleshooting images](#)
- [Troubleshooting containers](#)
- [Troubleshooting Docker Compose](#)
- [Troubleshooting data storage](#)
- [Known issues](#)
- [Getting help and support for IBM Cloud Container Service](#)

Debugging containers and reviewing logs

Review the options that you have to debug your containers and find the root causes for failures.

1. List your containers and review the current status.
 - `Single container`

```
bx ic ps -a
```

- Container group

```
bx ic groups
```

Optional: See [what each status means](#) and review tips to prevent a container from failing.

If your container failed or crashed, inspect your containers to find the root cause for the failure. The error message can be found in the **Failure** section of your response.

- Single container

```
bx ic inspect CONTAINER
```

- Container group

Inspect the container group.

```
bx ic group-inspect GROUP
```

For further information, list the instances of a container group and inspect each instance.

```
bx ic group-instances GROUP
```

```
bx ic inspect GROUP_INSTANCE
```

If the app exited with a failure, retrieve the logs for the app.

Note:

This command can be used only if the app writes logs to the standard STDOUT and STDERR output streams. Writing to these standard output streams allows you to view container logs even if the container shuts down or crashes.

- Single container

```
bx ic logs CONTAINER
```

- Container group

List the container instances of a container group.

```
bx ic group-instances GROUP
```

Retrieve the logs for each instance.

```
bx ic logs GROUP_INSTANCE
```

If you need to access logs of a container that is no longer available, such as an instance of a container group that has been re-created with autorecovery, use Kibana to view your logs.

`https://logmet.DomainName/`

Logs are available for up to 7 days in Kibana.

If the logs were saved into a file inside the container, you must log into the container to review it.

Important:

You can only log into a running container. If your container crashed and cannot be started again, you might not be able to access the logs of your container. Consider to change your app to write to the standard STDOUT and STDERR output streams so that you can access the log files even if the container is not accessible anymore.

If your container is not running, start your container.

```
bx ic start CONTAINER
```

Log into your running container.

```
bx ic exec -it CONTAINER bash
```

Navigate to your log file and review the logs.

If the logs were saved to a volume and your container cannot be started anymore, create a new container to access the volume.

Create a new container that is mounted to the volume where the log files are stored.

```
bx ic run --name logcontainer -m 64 --volume MOUNT_PATH_TO_VOLUME 1
```

Log into the running container.

```
bx ic exec -it logcontainer bash
```

Navigate to the volume mount path and review your log files.

If you need further support with debugging your containers, open a [support ticket](#).

Troubleshooting the CLI setup

Method error (rpc) during init

When you try to run `bx ic init`, you get one of the following messages.

```
bx ic rpc: can't find method CliRpcCmd.IsLoggedIn
```

```
bx ic rpc: can't find method CliRpcCmd.IsMinCliVersion
```

The installed version of the Cloud Foundry is an earlier version than what is supported for IBM Cloud Container Service.

Run `cf -v` to see which version of the Cloud Foundry CLI is installed.

If the installed version is earlier than 6.14.0, [update your Cloud Foundry CLI](#).

Registry authentication error received during `bx ic init`

When you run `bx ic init`, you see one of the following authentication error messages.

```
Could not authenticate with IBM Containers registry at registry.DomainName
****exit status 1
****time="2015-07-31T08:58:32-04:00" level=fatal msg="Post http://var/run/
unix /var/run/docker.sock: no such file or directory. Are you trying to c
```

```
** Retrieving client certificates from IBM Containers
Authentication failure. Check your username and password
Invalid token format. Please generate new token.
```

Either one of the `bx ic` prerequisites is not met or IBM Cloud Container Service is not available.

Verify that the Docker daemon is running. If it is not, start it and run `bx ic init` again.

Verify that you are logged in to the Cloud Foundry CLI by running `bx login`. Then, log in to the plug-in again by running `bx ic init`.

Verify the availability of IBM Cloud Container Service. [Check the Cloud status page for information](#).

Sudo is required with `bx ic` commands

When you run `bx ic` commands, you are required to use the `sudo` prefix with all of your commands.

By default, the `bx ic` commands must be run with root authentication and therefore, might require the prefix `sudo`, depending on your operating system.

To run `docker` and `bx ic` commands without `sudo`, run the following command, then, if you are on Linux, you must log out then log back in again.

```
sudo usermod -a -G docker <CURRENT_USER>
```

Troubleshooting quota

A container cannot be created because of insufficient org quota

When you create a container, you receive a message that you do not have sufficient quota in your space.

```
BXNUI0110E: Could not allocate IBM Containers resources. Insufficient ORG quota to new SPACE. Cannot exceed your organization 'organization' floating IPs. Use Cloud Manage Organizations to adjust quota allocation.
```

This message might be displayed when you try to create containers in a space that does not have enough quota allocated. By default, every space requires 2 IP addresses and 2 GB of memory to be allocated to that space to work with the container service.

To distribute the memory and IP quota across multiple spaces:

- From your account details, in manage organizations, select an organization.

- In the quota section, view the details for containers.

- Click **Edit** and allocate the memory and IP addresses across the spaces in the organization.

- Create a container in any space in the organization.

Quota cannot be edited though you have a pay account

You are not able to modify your container quota.

There might be inconsistencies between the organization's manager and the billing information.

The logged in user must be the manager of the organization. To verify the owner, complete these

steps.

From your account details, in manage organizations, select an organization.

In the user list, verify that the manager of the organization is the same as the ID that you are logging in with.

Log out of the account and log back in.

If you complete these steps and you still cannot edit container quota, contact IBM Support.

Troubleshooting images

Internal proxy blocks requests to push images

To create a container within a specific Cloud region, you must push the image to create the container to a registry within that region. When you try to push an image to that registry, the connection fails with one of the following error messages.

Connection refused

Connection timed out

You might have firewall settings that are preventing communication from your computer to the Cloud registry. Often, internal proxy settings are configured to allow only permissible communication for security reasons.

In your firewall, open communication from your computer to the registry for the region-specific IP addresses:

US South:

169.55.39.122

169.55.39.124

169.55.39.126

169.55.211.14

169.46.9.2

169.46.9.3

169.46.9.20

169.46.9.21

United Kingdom:

```
159.8.188.168
159.8.188.169
159.8.188.174
159.8.188.179
169.50.153.69
169.50.153.76
169.50.153.81
169.50.153.85
```

Problems with the IBM WebSphere Application Server Liberty images

You are having problems with using one of the IBM WebSphere® Application Server Liberty images in IBM Cloud Container Service.

The image might not be set up correctly.

If you have issues that relate specifically to this Dockerfile image, use the [GitHub issue tracker](#). For more general issues that relate to IBM WebSphere Application Server Liberty, you can get help by going to the [WASdev community](#).

For more information about the IBM WebSphere Application Server Liberty images in IBM Cloud Container Service, see [IBM WebSphere Application Server Liberty images](#).

Image cannot be pulled from your private Cloud images registry

You created a container from a container image in your private Cloud registry, but the container gets stuck in a **BUILDING** state. When you run the `bx ic inspect` command, the following error message is shown in the **Env** section of your CLI response.

```
"Env": [
    "metadata_exit_code=-1",
    "logging_password=",
    "space_id=0e65c436-9413-46e2-8b42-ff06aa74ac21",
    "logstash_target=logmet.opvis.bluemix.net:9091",
    "metadata_err_msg=Cannot pull image from registry: Pull failed w
to register layer: ApplyLayer exit status 1 stdout:  stderr: Co
cannot be mapped to a host ID",
```

```
"metrics_target=logmet.opvis.bluemix.net:9095",  
"metadata_running=1"
```

In the IBM Cloud Container Service, the [Docker user namespace feature](#) is enabled in the Docker Engine. The user namespace feature was first introduced with Docker version 1.10 and provides the possibility to map a root user in the container to a non-root user outside the container. In this way, the permissions of a user that can perform root actions inside the container, can be restricted outside the container, so files on the host and in other containers cannot be accessed, altered, or removed.

Note

The Docker user namespace is different to the namespace that you set up in Cloud to identify your [private images registry](#).

When you create a container from a container image with the `bx ic run` command, the image is loaded from your private Cloud images registry into the IBM Cloud Docker Engine. The Docker Engine then starts to create the respective container layers. Depending on what is specified in the container image, app files from your local machine might be added to the container. All these files are created by a user ID (UID) and group ID (GID) on your local machine. The UID and GID indicate the user's permission on the local machine. When you want to add these files to the container, the respective UID and GID is mapped to a UID and GID inside the container. The mapped UID and GID might have different permissions inside the container than on the local machine. For example, the local machine's non-root UID and GID is mapped to a root user inside the container. In this way, the user can act like a root user inside the container and access, alter, and remove container files. As the container is running in an isolated environment, the UID and GID cannot access any files on the host as it is assigned non-root permissions outside the container.

To map a local machine's UID and GID to one inside a container, the user namespace feature requires that all files are created with a UID and GID between 0 and 65536.

The error that is shown occurs when a file is created from a UID or GID outside this range and when this file is included in one of the container image layers. When you create a container from the image, the Docker Engine builds each layer of the image. When a file with a UID or GID

outside the range is found, the Docker Engine fails because this UID or GID could not be mapped to a valid one inside the container. The Docker Engine also fails when a file is used in one image layer and removed in the next image layer.

Enable the user namespace on your local Docker Engine

When you build your container images locally, be sure to enable the user namespace feature in your local Docker Engine. In this way, all UID, and GID on your local machine that are out of the valid user namespace range are automatically mapped to a valid ID within the range. Test your container image locally before you push it to your private Cloud images registry and use it with IBM Cloud Container Service. For more information about how to enable the user namespace feature, see [Starting the daemon with user namespaces enabled](#).

Comment out the affected image layer and change the ownership of the affected files

If you cannot find the file that causes the issue, try to comment out the image layers in your Dockerfile and uncomment one layer after another. Build your image and create a container to find the affected image layer.

Run `chown [-R] root:root <file_or_folder>` to change the ownership of the affected file that is stored on your local machine or add this command to the affected image layer in the Dockerfile.

Example Dockerfile

```
RUN curl -fsSL "https://www.kernel.org/pub/software/scm/git/git- $\{GIT\_VERSION\}$ " | tar -xzc /tmp \
    && cd /tmp/git- $\{GIT\_VERSION\}$  \
    && ./configure \
    && make all \
    && make install \
    && chown -R root:root /tmp
```

Image cannot be exported from your private Cloud images registry

You are logged in to the IBM Cloud Container Service and want to pull an image from your private Cloud registry to your local machine by using the `bx ic pull` command. The command fails stating that `bx ic pull` is not supported.

The `bx ic pull` command is not supported to pull an image from your private Cloud registry

to your local machine.

To export an image from your private Cloud registry, follow these steps.

[Log in to the IBM Cloud Container Service](#). If you are already logged in, make sure your environment variables are not set to use native Docker commands against the IBM Cloud Container Service.

List your images.

```
bx ic images
```

Pull an image from your private Cloud registry to your local machine. Use the name or id of the image that was returned in the previous step.

```
docker pull <image_name_or_id>
```

Verify that the image was pulled to your local machine.

```
docker images
```

Save your image into a .tar file.

```
docker export <image_name_or_id> > <exported_image_name>.tar
```

The namespace for an organization cannot be changed

You cannot find a method for changing the namespace.

The namespace for an organization cannot be changed after it is created.

You can create another organization and set the namespace for that new organization to the desired name.

Prompted for login during docker push

When you try to run `docker push` to add an image to your private Cloud images registry, you are prompted for login credentials. When you enter Docker or Cloud credentials, the authentication fails.

The authentication might fail due to the following reasons.

- The bearer token, which contains Cloud Foundry authentication information, is expired.

- The Cloud authorization service is not available.

If you are using the IBM Cloud Container Service plug-in:

Log in to the Cloud Foundry CLI by running the command to refresh the bearer token on your computer.

```
bx login [-a api.DomainName] [-sso]
```

The single-sign-on parameter `--sso` is required when you log in with a federated ID. If this option is used, open <https://login.ng.bluemix.net/UAALoginServerWAR/passcode> to obtain a one-time passcode. If you do not include the `--sso` option, complete these substeps.

For **Email**, enter the IBMId that you use to log in to Cloud.

For **Password**, enter the password for the IBMId that you use to log in to Cloud. By entering your user name and password, your organization and space are retrieved.

Enter the number that represents one of your Cloud organizations.

Enter the number that represents one of your existing Cloud spaces.

Initialize the IBM Cloud Container Service CLI to re-create your bearer token.

```
bx ic init
```

If the prompt still occurs after you logged in again, the Cloud authorization service is not available. Try again later.

Troubleshooting containers

502 Bad Gateway error when accessing an app

You deployed a container group successfully, but when you open the URL for the app, the following error is displayed.

```
502 Bad Gateway: Registered endpoint failed to handle the request.
```

The routing might still be setting up.

The container might not be responding.

There might be an error with the routing though the container group is running successfully.

Wait a few more minutes to allow enough time for the routing to complete. There might be a 5-minute wait after running the container process before the container group can connect to the specified route.

If you have waited 5 minutes and suspect that an error occurred, see if the container is responding by binding an IP address to the running container and then run a curl command to that IP address and port.

Run the following command to get the ID of one of the container instances in the group. Note the container ID and the port.

```
bx ic ps
```

Request an IP address by running the following command.

```
bx ic ip-request
```

Bind the IP address to the container instance by running the following command.

```
bx ic ip-bind IP_address container_ID
```

In a browser, enter the URL to the application with the IP address and the port.

Example: `http://IP_address:port`

If you cannot access the application, then there is a problem with the container group and the group must be re-created. If you can access the application, the container is running successfully, but there is an error with the routing. Contact IBM Cloud Support.

Internal proxy blocks requests to the service

When you try to access IBM Cloud Container Service, the connection fails with one of the following error messages.

```
Connection refused
```

```
Connection timed out
```

You might have firewall settings that are preventing communication from your computer to IBM

Cloud Container Service. Often, internal proxy settings are configured to allow only permissible communication for security reasons.

In your firewall, open communication from your computer to IBM Cloud Container Service for the region-specific IP addresses:

US South:

```
169.46.9.7
169.46.9.19
169.46.9.24
169.46.9.28
169.55.211.15
169.55.211.17
169.55.211.21
169.55.211.28
```

London:

```
159.8.188.181
159.8.188.182
159.8.188.184
159.8.188.188
169.50.153.66
169.50.153.77
169.50.153.91
169.50.153.94
```

Container group instances are repeatedly re-created during startup by the IBM Cloud Container Service auto-recovery mode

You created a container group and enabled the auto-recovery. During the startup of the container group, unresponsive group instances are removed and re-created repeatedly. The app that is running in your container group takes too long to start listening on the exposed port. If you enabled auto-recovery for your container group, IBM Cloud Container Service checks the health of each instance with an HTTP request on the exposed TCP port after the group is created. If an instance is not listening on the exposed port within 3 minutes, the instance is marked unresponsive and is removed, re-created, and restarted. This process

repeats until the maximum number of re-created instances is reached. Then, auto-recovery is disabled.

To run an app in a container group with auto-recovery enabled, make sure that the app starts and listens on the exposed port in less than 2 minutes. Depending on the type of app, different methods exist to ensure a quick startup of your app.

Cloud service does not bind directly to a container

Your Cloud service does not bind directly to a container by using service binding.

Some Cloud services do not support direct service binding to a container and lead to a failure during container creation. For these services, service binding does not pass along the environment variable `VCAP_SERVICES` to the container. To work around this problem, you can use the CLI to create a Cloud Foundry app that acts as a bridge for sending `VCAP_SERVICES` from the service to a container.

Select a space in which to deploy a container, such as `dev`.

Identify an existing app or create a Cloud Foundry app to bind the Cloud service to. This app is used for the passing of the `VCAP_SERVICES` information, so it is referred to as a bridge app.

To create a Cloud Foundry app:

Tip

The app that you create by completing these steps is a placeholder for storing the `VCAP_SERVICES` services. To incur fewer costs than a running app, this app does not start or run. The app is not required to run to provide the `VCAP_SERVICES` to the container.

Create a directory and a text file in your working app directory. Example:

`<directory>/<empty_file.txt>`. This file serves as the source file for the app, though it does not need to contain any content.

- Windows example: `C:\Users\User\containerbridge\empty_file.txt`

- Linux example: `/user_home_directory/containerbridge/empty_file.txt`
- OS X example: `/Users/user_home_directory/containerbridge/empty_file.txt`

Run this command to create a zero-runtime app at no costs. The app can be named anything though *containerbridge* is a common name since the app acts a bridge from the container to a service.

```
cf push <appname> -p <directory> -i 1 -d AppDomainName -k 1M -m 64M
```

Example:

```
cf push containerbridge -p containerbridge -i 1 -d AppDomainName -k 1M -m 64M
```

If the Cloud service you want to bind isn't already added, add the service to the Cloud space.

To get a list of the available Cloud services, run the `cf marketplace` command. The resulting list includes the name of the Cloud service, its plan type, and a description of what the service is.

Run the `create-service` command and include the service name and plan from the `cf marketplace` list. Also, include your own instance name for the service.

```
cf create-service <service> <plan> <service_instance>
```

Example

```
cf create-service cleardb spark my-clear-db
```

Bind the Cloud service to the app by running the `bind-service` command. Repeat to bind other Cloud services to the bridge app.

```
cf bind-service <appname> <service_instance>
```

Example

```
cf bind-service containerbridge my-clear-db
```

If you are prompted to restage the app, run the following command.

```
cf restage containerbridge
```

Review the information for the Cloud service that is passed through the *containerbridge* app by running the following command.

```
cf env <appname>
```

Example

```
cf env containerbridge
```

The results include the injected Cloud service parameters by using the VCAP_SERVICES environment variable.

For example:

```
{
  "VCAP_SERVICES": {
    "<service_name>": [
      {
        "credentials": {
          "hostname": "<hostname_id>",
          "jdbcUrl": "<jdbc_URL>",
          "name": "<encoded_name>",
          "password": "<encoded_password>",
          "port": "<port_number>",
          "uri": "<service_URI>",
          "username": "<encoded_username>"
        },
        "label": "<service_name>",
        "name": "<service_instance>",
        "plan": "<plan_name>",
        "tags": [
          "<tag1>",
          "<tag2>",
          "<tag3>"
        ]
      }
    ]
  }
}
```

After you create the binding between the Cloud services and the *containerbridge* app, create a container and bind the *containerbridge* app to it. For more information, see the CCS_BIND_APP flag in the environment variables in [Creating a single container](#) or [Creating a container group](#).

Tip

After the bridge app is bound to a container, do not delete the bridge app. If you delete a bound bridge app, the container hangs. If you no longer require the bridge app, delete the container and any other containers that are using the bridge app before you delete the bridge app.

Cannot create child process with su command

When using the substitute user (su) Linux command to edit the users for a container, you receive the message `su: cannot create child process: Resource temporarily unavailable`, even though the su command worked in the local Docker environment. You might experience this issue when you log into a container or when the su command is run as part of a script within the container.

Example:

```
bx ic exec -it my_container bash
```

```
adduser user
```

```
su - user
```

Result:

```
su: cannot create child process: Resource temporarily unavailable
```

This issue often occurs in containers that are built with Centos 7 images, though it might occur with other image types as well.

You can resolve the issue by editing the Dockerfile and re-creating the image or by logging in to the running container.

To resolve the issue by editing the Dockerfile:

In the local directory you are creating your image from, create an `etc` folder, if one does not exist already.

```
mkdir etc
```

Navigate to the `etc` directory.

```
cd etc
```

In the `etc` folder, create a `pam.d` directory.

```
mkdir pam.d
```

Navigate to the `pam.d` directory.

```
cd pam.d
```

In the `pam.d` directory, create a file named `su` and open it with a text editor.

In the `su` file, copy in the following text where all of the session variables are set to `optional` and save the changes to the file.

```
##%PAM-1.0
auth            sufficient      pam_rootok.so
# Uncomment the following line to implicitly trust users in the "wheel" g
#auth          sufficient      pam_wheel.so trust use_uid
# Uncomment the following line to require a user to be in the "wheel" gro
#auth          required        pam_wheel.so use_uid
auth           substack        system-auth
auth           include         postlogin
account        sufficient      pam_succeed_if.so uid = 0 use_uid quiet
account        include         system-auth
password       include         system-auth
session        optional        system-auth
session        optional        postlogin
session        optional        pam_xauth.so
```

In the Dockerfile for the image, include the following information.

```
RUN sed -i 's/session.*include/session        optional/' /etc/pam.d/su
```

Re-create the image. For more information, see [Building an image](#).

Re-create the single container or container group. For more information, see [Creating a single container](#) or [Creating a container group](#).

To verify, run the following command.

```
bx ic logs container_name
```

Result:

```
%Changing password for user user_name.  
passwd: all authentication tokens updated successfully.
```

To resolve the issue by logging in to the running container:

Create the container.

Log into the container.

```
bx ic exec -it my_container bash
```

Navigate to the etc directory.

```
cd etc
```

Navigate to the pam.d directory.

```
cd pam.d
```

Open the visual editor by using the Linux vi command.

```
vi su
```

In the su file, change all of the session variables to optional and save the changes to the file. For more information about using the vi editor, see [How to Use the vi Editor](#).

```
##%PAM-1.0  
auth            sufficient      pam_rootok.so  
# Uncomment the following line to implicitly trust users in the "wheel" group  
#auth           sufficient      pam_wheel.so trust use_uid  
# Uncomment the following line to require a user to be in the "wheel" group  
#auth           required        pam_wheel.so use_uid  
auth            substack        system-auth  
auth            include         postlogin  
account         sufficient      pam_succeed_if.so uid = 0 use_uid quiet  
account         include         system-auth  
password        include         system-auth
```


session	optional	system-auth
session	optional	postlogin
session	optional	pam_xauth.so

Log in as the user.

```
su - user
```

Run the `id` Linux command to see the configuration information for the user.

```
id
```

Result:

```
uid=1000(user) gid=1000(user) groups=1000(user)
```

Container stops and shuts down after creation

You created a container, but it shut down immediately.

You ran a container locally and it ran successfully, but when you ran the container in Cloud, you see one of the following situations.

From the Cloud dashboard, the container state changes from Running to Stopped.

From the CLI, when you run `bx ic inspect container`, you see that the state status is Shutdown.

You also tried running `bx ic logs`, but did not find any issues with the container.

When you run a container in Cloud, the container is always run detached mode, similar to running `docker run -d` when a container is run locally. As a result, even containers that start locally do not run continuously in Cloud.

You must make the container a long-running application by editing the Dockerfile. For example, if you added a command to output the phrase "hello world" every ten seconds, that continuous action makes your container a long-running application.

```
ENTRYPOINT ["/bin/sh"]  
CMD ["-c","while true; do echo hello world; sleep 10;done"]
```

Deleted container continues to show in the Cloud GUI


From the Cloud GUI, you deleted a container, but the container continues to be displayed in the Cloud GUI and uses quota.

From the Cloud GUI, the deletion could not be completed.

Delete the container from the command line.

For single containers, follow these steps.

Delete the container by running one of the following command line options.


-  IBM Cloud Container Service plug-in with `bx ic` commands

```
bx ic rm [-f] CONTAINER [CONTAINER]
```

-  IBM Cloud Container Service plug-in with `docker` commands

```
docker rm [-f] CONTAINER
```

Verify that the container was removed by running one of the following commands to list all of the containers in your space:

-  IBM Cloud Container Service plug-in with `bx ic` commands



```
bx ic ps -a
```

-  IBM Cloud Container Service plug-in with `docker` commands

```
docker ps -a
```

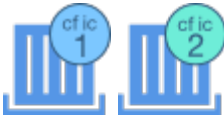
For container groups, follow these steps.

Delete the container group by running the following command.

-   IBM Cloud Container Service plug-in

```
bx ic group-remove [-f] GROUP [GROUP]
```

Verify that the container was removed by running the command to list all of the containers in your space.

- o  IBM Cloud Container Service plug-in

```
bx ic groups [-q]
```

If you run the commands and the container continues to display in your Cloud GUI, wait to see whether it gets removed later.

Container group is showing a **HAS_INACTIVE_INSTANCES** status

When you run `bx ic group-inspect`, the container group is displaying a `HAS_INACTIVE_INSTANCES` status.

If the networking fails to be configured when the group is created or fails the health check for one or more instances, the instance state is change to `INACTIVE`. If any instance in a group is inactive, the group state is changed to `HAS_INACTIVE_INSTANCES`.

If the issue occurs when the health check fails for an instance, the issue can resolve without any action required. If the issue occurs when the group is first created, you might try re-creating the group. If the issue occurs again after re-creating, networking might not be configurable for the app when it is deployed as a group. Try deploying the app as a single container instead.

Troubleshooting Docker Compose

Insecure HTTPS connection warning is displayed when executing Docker Compose commands

When you run Docker Compose commands in the CLI, a warning similar to one of the following examples is displayed. Other related error messages can be found in the [urllib3 development documentation](#).

```
/.../requests/packages/urllib3/util/ssl_.py:315: SNIMissingWarning: An HTTPS
has been made, but the SNI (Subject Name Indication) extension to TLS is not
platform. This may cause the server to present an incorrect TLS certificate,
validation failures. For more information, see
https://urllib3.readthedocs.org/en/latest/security.html#snimissingwarning. SN
/.../requests/packages/urllib3/util/ssl_.py:120: InsecurePlatformWarning: A t
SSLContext object is not available. This prevents urllib3 from configuring SS
and may cause certain SSL connections to fail. For more information, see
https://urllib3.readthedocs.org/en/latest/security.html#insecureplatformwarn:
```

This error message is typical for Linux operating systems and is caused by a missing verification of HTTPS requests.

Option 1: Install the `ndg-httpsclient` to avoid SSL related error messages and warnings.

```
pip install ndg-httpsclient==0.4.0
```

Option 2: If the `ndg-httpsclient` is already installed on your computer, upgrade the package to the latest version.

```
pip install --upgrade ndg-httpsclient
```

No module named queue error is displayed when executing Docker Compose commands

As soon as you run Docker Compose commands in your CLI, an error message similar to the following example is displayed.

```
File "/usr/local/bin/docker-compose", line 9, in <module>
load_entry_point('docker-compose==1.6.2', 'console_scripts', 'docker-compose')
"/System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python/p
in load_entry_point return get_distribution(dist).load_entry_point(group, nam
"/System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python/p
in load_entry_point return ep.load() File "/System/Library/Frameworks/Python.
line 2108, in load entry = __import__(self.module_name, globals(),globals(),
File "/Library/Python/2.7/site-packages/compose/cli/main.py", line 26, in <mo
File "/Library/Python/2.7/site-packages/compose/project.py", line 10, in <mo
File "/Library/Python/2.7/site-packages/compose/parallel.py", line 9, in <mo
No module named queue
```

The Python path is pointing to the wrong directory. The Docker Compose Python package cannot be found.

Change the Python path to the correct directory.

```
export PYTHONPATH=/Library/Python/2.7/site-packages/:$PYTHONPATH
```

Troubleshooting data storage

Permission error changing the owner of a mounted volume directory in a container

You get a permission error when you try to change the ownership of a file or directory for a volume that is mounted in a container.

For IBM Cloud Container Service, the [user namespace feature](#) is enabled for Docker Engine. The effective root inside the container is mapped to a non-root user on the compute host. The source of the mounted volume is a network file system (NFS). Since the NFS is not aware of user namespaces, all non-user checks are performed for the operations on mounted files and directories inside the container.

In the Dockerfile for an image, add the following lines to create a user with an available user ID (UID). For example, you can name the user *volumeeditor*.

```
RUN groupadd --gid 1010 volumeeditor
RUN useradd --uid 1010 --gid 1010 -m --shell /bin/bash volumeeditor
```

Build and push the image to your private Cloud images registry.

Create a container with the image where the volume path is mounted.

Tip:

Run `bx ic namespace-get` to retrieve your namespace and replace `<my_namespace>` with your namespace information.

```
bx ic run --volume volume_name:/path/to/files --name fixVolumePermissions
```

Log in to the container.

```
bx ic exec -it container_name bash
```

Define an environment variable that is named MOUNTPATH for the mount path.

```
MOUNTPATH="/path/to/files"
```

Add the user to the group root.

```
adduser volumeeditor root
```

Set permissions so that all group members have read and write access to the volume mount.

```
chmod 775 $MOUNTPATH
```

Create a folder in the mountpath and set the read and write permissions. In this example the folder is named *data*.

```
su -c "mkdir -p $MOUNTPATH/data" volumeeditor
```

```
su -c "chmod 700 $MOUNTPATH/data" volumeeditor
```

Verify that the user *volumeeditor* can access the files and directories in the mount path.

```
sudo -u volumeeditor ls -al $MOUNTPATH
```

Optional: Link a folder in the container to the *data* folder in the volume. This link associates the write permissions of `$MOUNTPATH/data` with any folder in the container.

```
ln -sf $MOUNTPATH/data /folder/in/container/
```

If the edits can be made without a permission error, remove the *volumeeditor* group root user.

```
deluser volumeeditor root
```

Logs from container group instances are overwritten in a volume

When you created a container group, you mounted a volume and have logs outputted from each container group instance to the volume. Each container group instance writes to the same files and the files are overwritten by each of the instances.

Example:

```
bx ic group-create -p 9080 -m 64 -v log_vol:/opt/ibm/logs -e LOG_LOCATIONS=
```

Each container instance is writing a file to the volume as if it were the sole writer. Each instance has a local cache in memory. The files in the volume are repeatedly replaced by the cached data from each container instance.

Adjust the name of the log file to include an identifier so that the files are different for each instance. For example, include the `$HOSTNAME` variable in the log file name that is generated by the app and specified when the container group is created.

Example:

```
bx ic group-create -p 9080 -m 64 -v log_vol:/opt/ibm/logs -e LOG_LOCATIONS=
```

Transferring data to or from a volume takes longer than expected

Reading to or writing from a volume takes longer than expected.

In IBM Cloud Container Service, volumes do not have access to the local host. Therefore, all volumes are mounted on a NFS file share. The performance of the volume is dictated by the size and IOPS (input/output per second) of your file share, along with the work load of your application. The file share your volume is hosted on may not be suited for the workload of your volumes and containers. In addition, multiple volumes on one file share can compete for the file share's resources.

To create a new file share, you must have the organization manager role. For detailed information, see [Users and roles](#).

Determine the ID of the file share that your volume is hosted on.

```
bx ic volume-inspect my_volume
```

```
{
```

```
"fsId": "fa277ff4-8a64-435b-9b75-0f11d59a3ae4",
"hostPath": "/vol/fa277ff4-8a64-435b-9b75-0f11d59a3ae4/my_volume",
"otherSpaceVisibility": [],
"spaceGuid": "fa277ff4-8a64-435b-9b75-0f11d59a3ae4",
"volName": "my_volume"
}
```

View your file share to determine its size and IOPS.

```
bx ic volume-fs
```

Name	Size	IOPS/GB	Created
<i>fa277ff4-8a64-435b-9b75-0f11d59a3ae4</i>	20	4.00	2016-05-20 13:48:

View the available sizes in GB for your file shares.

```
bx ic volume fs-flavor
```

```
[
  20,
  40,
  80,
  100,
  250,
  500,
  1000,
  2000,
  4000,
  8000,
  12000
]
```

Determine the size and IOPS that you need for your apps' workload. Valid IOPS values are 0.25, 2 or 4 IOPS per GB. The file share size and IOPS per GB multiply together to determine the file share's total IOPS. This table includes examples of how different file share sizes at 4 IOPS per GB scale.

Table 1. Examples of file share sizes and IOPS per GB

File share sizes	IOPS per GB	Total IOPS
20 GB	4	80
100 GB	4	400
1,000 GB	4	4,000
4,000 GB	4	16,000

Create a new file share. The first number is the size in GB. The second numbers is the file share IOPS.

```
bx ic volume-fs-create my_new_fs 1000 4
```

Verify the new file share and wait for the status to show as ready.

```
bx ic volume-fs
```

Create your volume on the new file share.

```
bx ic volume-create my_new_volume my_new_fs
```

Run a container or container group from your image with the new volume mounted.

Back up data from *my_volume* and copy the data to *my_new_volume*. See [Backing up and restoring data](#) for more information.

Parsing error when creating a container with a volume from the CLI

When you create a single container or container group from the CLI, you receive a failed message.

Example

```
bx ic group-create --name my_container -n my_host -d AppDomainName -m 2048 -
```

Result

```
docker: Error parsing reference: "my_volume:/my/path" is not a valid reposit
```

The CLI for Cloud Foundry version 6.17 and later uses the `-v` parameter for another purpose. In your command, change the `-v` option to the `--volume` option.

Example

```
bx ic group-create --name my_container -n my_host -d AppDomainName -m 2048 -
```

Known issues

Learn about the known issues that occur in IBM Containers.

Docker and IBM Cloud Container Service

Some Docker commands are not supported in IBM Cloud Container Service plug-in (bx ic)

Only certain Docker commands are supported in (bx ic). You can view a full list of supported commands in the [docker command reference](#). For example, `tag` is not available in `bx ic`, but `docker tag` can be used locally and then the updated image can be pushed to the Cloud registry.

Common issues between Docker and Windows

Common issues for installing and configuring Docker on Windows computers exist. For more information about these common issues, review [Blog: Installing Docker for Windows: Fixes for common problems](#) and [Stack Overflow posts tagged with boot2docker](#).

Images

The pushing and pulling of large images fails

When you run a Docker command to push or pull a large image from your organization's private Cloud registry, the request fails. To determine whether an issue with the image exists, first verify that the image can be pushed to DockerHub. If no issue exists with the image, if possible, reduce the size of the image.

An image's size in Cloud does match its size locally

When you push an image to your private Cloud registry, the size reported for the

image is smaller than the size of the same image in your local Docker engine. The difference between the sizes does not indicate that an issue occurred when the image was pushed. The compressed size of the image is reported in IBM Cloud Container Service.

Monitoring and logging

Extra characters in the output for `bx ic logs`

When you run `bx ic logs`, extra characters, such as `&`, are displayed at the beginning of each line. This issue occurs when incoming docker logs are parsed by NGINX.

Log file additions cannot be specified with expressions

You cannot use filters or wildcard expressions for specifying the log files that are to be collected. For example, you cannot specify `/var/log/*.log`. Instead, list each log file individually.

Single containers and container groups

Failed connection after you created a container

Due to a limitation in the Docker networking architecture of IBM Cloud Container Service, outbound network access might not work at initial start up of the container. If your container attempts to make an outbound network connection to return an error code or to throw an exception if the network is not available, the container exits and a "Crashed" state is displayed.

To allow the networking to initialize and to enable the container deployment to proceed, you must have a sleep statement before you start any processes that require outbound connectivity. Create a custom Dockerfile to wrap any existing server startup commands in the base Dockerfile. For an example, see the [Let's Chat sample application](#).

Inbound connectivity is not immediate

Inbound connectivity is not immediate for both container groups and single containers, including routes and external apps. Wait for the connectivity to become

available, which can take up to 5 minutes.

Different default units of measurement used in `docker run -m` and `bx ic run -m`

When no unit of measurement is specified with `docker run -m`, bytes is used as the default unit of measurement. When no unit of measurement is specified with `bx ic run -m`, MB is used as the default unit of measurement.

User data limit prevents a container from being created

If you have too many services bound to a container, it cannot be created. Each service injects data into the container and is counted toward a 64KB limit. When you create a container that exceeds this limit, you receive a message, `Resource CREATE failed... User data too large`. If you see this message, remove a service from your container.

Port 25 is not available to expose in a container

Port 25 is blocked from use by SoftLayer, which is part of the IBM Cloud Container Service architecture. This issue affects the usage of some SMTP services in containers.

Container building or networking process does not complete

If a container remains in the building or networking state too long, you might need to remove the container and create a new container from the same image. You can use the following CLI `bx ic` commands:

Retrieve a list of containers that are running in the space. By default, only running containers are displayed. For more information, review the [CLI `bx ic ps` reference](#).

```
bx ic ps -a
```

Remove one or more containers. For more information, review the [CLI `bx ic rm` reference](#).

```
bx ic rm -f container_ID
```

Start a new container in the container cloud service from an image name. For more information, review the [CLI `bx ic run` reference](#).

```
bx ic run
```

Auto-recovery is not immediate for container groups

Auto-recovery for container groups might take more than 15 minutes for new systems

to come online. Wait for auto-recovery to become available, which can take more than 15 minutes.

cf delete-orphaned-routes command deletes routes from container groups

Cloud Foundry does not recognize routes that are mapped to container groups in the same way that it recognizes routes as bound to applications. Do not run the `cf delete-orphaned-routes` command when you use container groups.

Getting help and support for IBM Cloud Container Service

Where do you start troubleshooting a container?

To see whether Cloud is available, [check the Cloud status page](#).

Review the forums to see whether other users ran into the same issue. When you use the forums to ask a question, tag your question so that it is seen by the Cloud development teams.

- If you have technical questions about developing or deploying an app with IBM Cloud Container Service, post your question on [Stack Overflow](#) and tag your question with `ibm-bluemix` and `containers`.
- For questions about the service and getting started instructions, use the [IBM developerWorks dW Answers](#) forum. Include the `bluemix` and `containers` tags.

See [Getting help](#) for more details about using the forums.

Contact IBM Support. For information about opening an IBM support ticket, or about support levels and ticket severities, see [Contacting support](#).