

# On-Demand Services SDK

## Documentation


Version 0.16

Published: 12 Nov 2018

## Table of Contents

Table of Contents	2
On-Demand Services SDK	3
On-Demand Services SDK Release Notes	5
v0.16.1 Release Notes	5
Known Issues	5
About the On-Demand Services SDK	6
Getting Started: ODB on a Local Development Environment	9
Creating the Service Author Deliverables	15
Operating an On-Demand Broker	27
Backup and Restore Considerations	43
Troubleshooting Errors	44
Troubleshooting Components	47
Techniques for Troubleshooting	50
Creating an On-Demand Service Tile	56
How On-Demand Services Process Commands	60
Frequently asked questions	63

## On-Demand Services SDK

 **Note:** On-Demand Services SDK v0.16 does not support any of the currently supported versions of Ops Manager. For your product to stay up-to-date with the latest software, features, and security updates, use the latest version of On-Demand Services SDK.

This guide is intended for people who want to author service tiles for Pivotal Cloud Foundry (PCF) using the on-demand services SDK, part of the Pivotal Cloud Foundry Services SDK.

## Overview

PCF operators make software services such as databases available to developers by using the Ops Manager **Installation Dashboard** to install service tiles. Before BOSH 2.0, operators configured a service tile by pre-assigning a block of VMs with fixed CPU, hard disk, and RAM levels to allocate as instances for each service. This limited the possible number of instances and demanded wasteful one-size-fits-all resource provisioning.

On-demand services let you provision instances more flexibly. The operator does not pre-allocate a block of VMs for the instance pool, and they can specify an allowable range rather than fixed settings for instance resource levels. When a developer creates an on-demand service instance, they then provision it at creation time.

The on-demand services SDK simplifies broker and tile authoring, and is the standard approach for both Pivotal internal services teams and Pivotal partner independent software vendors (ISVs) to develop on-demand services for PCF.

The [About the On-Demand Services SDK](#) topic describes in greater detail how the on-demand broker works within PCF.

## Product Snapshot

Current On-Demand Services SDK details:

Version: 0.16.1  
 Release date: 07/06/17  
 Compatible Ops Manager version(s): v1.9, v1.10, v1.11  
 Compatible Elastic Runtime version(s): v1.9, v1.10, v1.11  
 vSphere support? Yes  
 AWS support? Yes  
 GCP support? Yes  
 Azure support? Yes  
 OpenStack support? Yes

## Key Features

The benefits of provisioning IaaS resources on-demand are:

- Scale resource consumption linearly with need, without having to plan for pre-provisioning.
- App developers get more control over resources, and do not have to do acquire them through the operator.

The benefits of using ODB to develop on-demand services are:

- ODB reduces the amount of code service developers have to write by abstracting away functionality common to most single-tenant on-demand service brokers.
- ODB uses BOSH to deploy service instances, so anything that is BOSH-deployable can integrate with Cloud Foundry's services marketplace.

ODB uses the following BOSH features:

- Dynamic IP management
- Availability zones
- Globally-defined resources ( **Cloud Config**). This results in manifests that are portable across BOSH CPIs, and are substantially smaller than old-style manifests.
- Links between deployed BOSH instances consuming information, e.g. IP addresses, of other instances.

## Prerequisites for deploying brokers that use ODB

Minimum versions of Cloud Foundry and BOSH are described in [the operator section](#).

## On-Demand Services SDK Release Notes

### v0.16.1 Release Notes

#### Minimum Version Requirements

- BOSH 257+ (261+ for lifecycle errands) / BOSH lite v9000.131.0
- CF 238+

#### Known Issues

💡 On PCF for GCP deployments, Ops Manager service network VMs are not assigned the correct firewall rules. As a result, these VMs cannot communicate with the BOSH Director and service tiles that use the On-Demand Service Broker (ODB) fail to create service instances. As a workaround, if you are deploying a service network in GCP, modify your firewall to use subnet CIDR-based rules.

💡 On Demand Broker log files in ``/var/vcap/sys/log/broker/*`` will be truncated when the broker process restarts via Monit. If you have syslog forwarding configured there should be no impact. Recommendation is to upgrade to ODB 0.17.0 to resolve this issue.

#### New Features

#### Service Instance Lifecycle Errands

- A new errand can be specified in manifests that will delete all service instances and deregister the broker, called `delete-all-service-instances-and-deregister-broker`
  - This will decrease uninstall times for tiles.
  - At the start of this new errand, service access is disabled so no more instances can be created.
  - Before this release, two separate errands had to be specified: `deregister-broker` and `delete-all-service-instances`.

#### Other Enhancements

- Broker API supports custom response codes and error messages for create/update/bind/unbind/delete. syslog-migration
- Use Golang 1.8.3 in the ODB BOSH Release

#### Breaking Changes

No breaking changes.

#### Bug fixes

- Updating a service instance will now return an HTTP 422 error when there is a pending change on the service instance.
- The `upgrade-all-service-instances` errand now exits with a non-zero status code if upgrading any instances fails.
- When co-locating [syslog-migration-release](#) or [syslog-release](#) there would be duplicate log entries, as ODB was also writing its logs to syslog. Now ODB only writes its logs to `/var/vcap/sys/log`, and anything syslog related is handled by syslog-release or syslog-migration-release. [0.16.1]

## About the On-Demand Services SDK

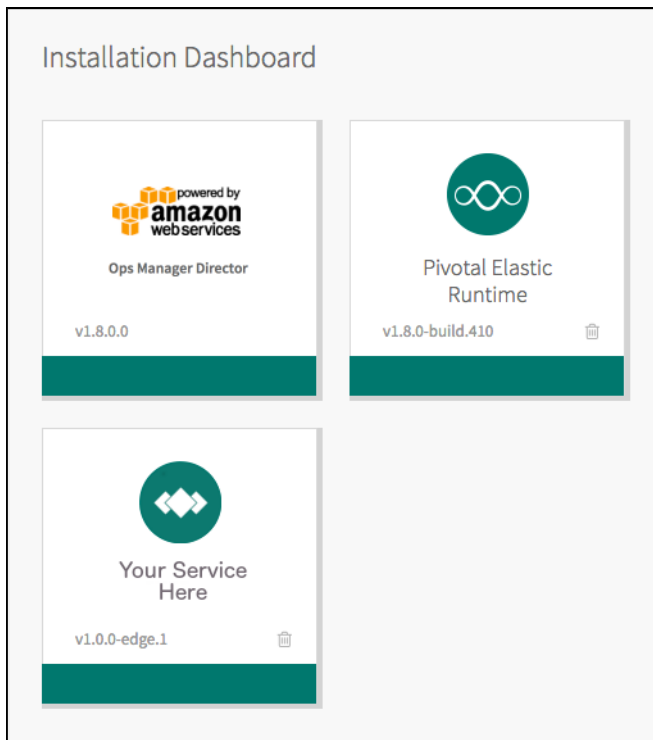
### Cloud Foundry Service Brokers and PCF Tiles

Service brokers let developers create service instances in their development spaces that they can call from their code. To do this, the brokers provide an interface between the Cloud Controller and the add-on software service that they represent. The service can run internal or external to a CF deployment, but the service broker always runs inside the cloud.

The service broker works by providing an API which the Cloud Controller calls to create service instances, bind them to apps, and perform other operations. Cloud Foundry service brokers are implemented as HTTP servers that conform to the [service broker API](#).

In addition to providing an API, a service broker publishes a service catalog that may include multiple service plans, such as a free tier and a metered tier. Brokers register their service plans with the Cloud Controller to populate the services marketplace, which developers access with `cf marketplace` or through the Pivotal Cloud Foundry (PCF) Apps Manager.

On PCF, cloud operators make software services available to developers by finding them on [Pivotal Network](#) and then installing and configuring them through a tile interface in the Ops Manager **Installation Dashboard**. Installing a service tile creates a service broker, registers it with the Cloud Controller, and publishes the service plans that the broker offers. Developers can then create service instances in their spaces and bind them to their apps.



The central element behind a service tile is the service broker, but the tile software includes other components that make the service easy for operators to install and maintain, and easy for developers to use. These components include configuration layouts, upgrade rules, lifecycle errands, and BOSH manifests for deploying the service instances.

### BOSH 2.0 and Service Networks

Before BOSH 2.0, cloud operators pre-provisioned service instances from Ops Manager. In the Ops Manager Director **Networking** pane, they allocated a block of IP addresses for the service instance pool, and under **Resource Config** they provisioned pool VM resources, specifying the CPU, hard disk, and RAM they would use. All instances had to be provisioned at the same level. With each `create-service` request from a developer, Ops Manager handed out a static IP address from this block, and with each `delete-service` it cleaned up the VM and returned it to the available pool.

With BOSH 2.0 dynamic networking and Cloud Foundry asynchronous service provisioning, operators can now define a dynamically-provisioned service network that hosts instances more flexibly. The service network runs separate from the PCF default network. While the default network hosts VMs launched by Ops Manager, the VMs running in the service network are created and provisioned on-demand by BOSH, and BOSH lets the IaaS assign IP addresses to the service instance VMs. Each dynamic network attached to a job instance is typically represented as its own Network Interface Controller in

the IaaS layer.

Operators enable on-demand services when they deploy PCF, by creating one or more service networks in the Ops Manager Director **Create Networks** pane and selecting the **Service Network** checkbox. Designating a network as a service network prevents Ops Manager from creating VMs in the network, leaving instance creation to the underlying BOSH.

Name\*
A unique name for this network

☒ Service Network

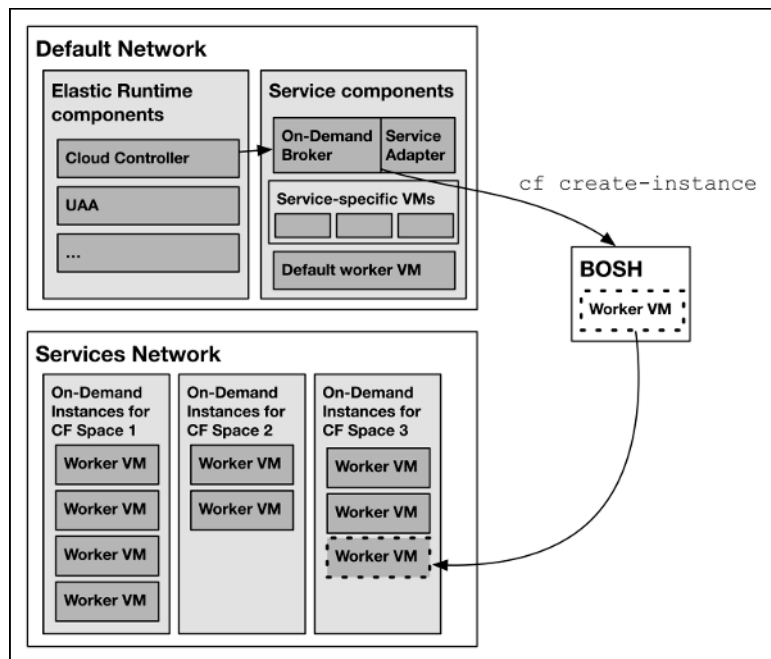
When they deploy an on-demand service, operators select the service network when configuring the tile for that on-demand service.

## On-Demand Services

On-demand services use the dynamically-provisioned service network to host the single-tenant worker VMs that run as service instances within development spaces. This architecture lets developers provision IaaS resources for their service instances at creation time, rather than the operator pre-provisioning a fixed quantity of IaaS resources when they deploy the service broker.

By making services single-tenant, where each instance runs on a dedicated VM rather than sharing VMs with unrelated processes, on-demand services eliminate the “noisy neighbor” problem when one application hogs resources on a shared cluster. Single-tenant services can also support regulatory compliance where sensitive data must be compartmentalized across separate machines.

An on-demand service splits its operations between the default network and the service network. Shared components of the service, such as executive controllers and databases, run centrally on the default network along with the Cloud Controller, UAA, and other PCF components. The worker pool deployed to specific spaces runs on the service network.



## On-Demand Services SDK and the On-Demand Broker

The on-demand services SDK is an SDK you can use to create on-demand brokers for single-tenant service offerings.

The on-demand services SDK simplifies broker and tile authoring, and is the standard approach for both Pivotal internal services teams and Pivotal partner independent software vendors (ISVs) to develop on-demand services for PCF.

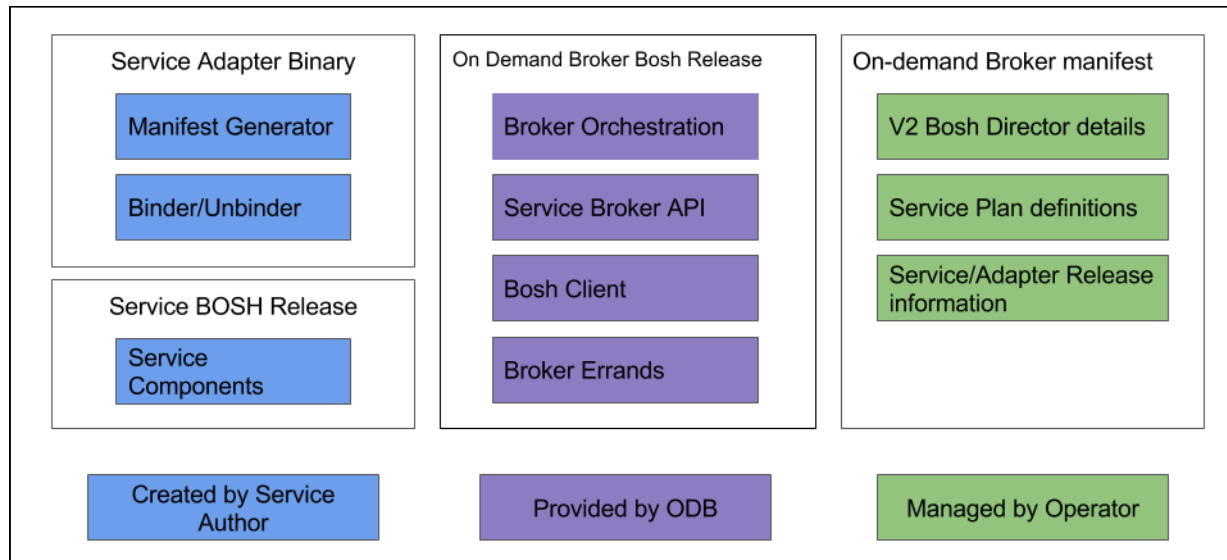
The ODB SDK provides a generic on-demand broker (ODB) that answers API calls from the Cloud Controller. The service author plugs service-specific functionality into the ODB SDK via an executable called a **Service Adapter**. For more information about the responsibilities of service authors, please see [Creating the Service Author Deliverables](#).

No additional or third-party components other than the service broker and the BOSH release for the service itself are required. This simplifies the setup. Everything is done through the single install process. This approach also simplifies support because there are fewer moving parts, and your customer's network needs less customizing of DNS rules and additional firewall ports.

The on-demand services SDK imposes no constraints on the service authors' ability to offer new functionality or expose configuration options in their service plans, such as rate limiting and external load balancers.

## Service Adapters

A service adapter is a binary that is called out by the ODB when it wants to do service-specific tasks.



The above diagram shows where responsibility lies for each aspect of the ODB workflow.

The service author can focus on building the BOSH release of their service and provide a service adapter binary that manages manifest generation, binding, and unbinding. The ODB manages all interactions with Cloud Foundry and BOSH.

Thanks to BOSH v2, service authors can define resources globally (in **Cloud Config**). This makes manifests portable across BOSH CPIs and lets them be substantially smaller than old-style manifests. The ODB takes advantage of other BOSH v2 features as well, including dynamic IP management, availability zones, and links through which deployed BOSH instances can access IP addresses and other information from other instances.

Once an on-demand tile is authored and distributed, the operator installs and configures it the same way they do with any other Pivotal products. In the process, they select which of the tile's available service plans to offer their developers.

## Procedures for Using ODB

The following procedures outline how to set up, create and maintain a service tile based on the ODB SDK:

- [Setting up your BOSH director](#) — The operators ensure that minimum versions of Cloud Foundry and BOSH are available.
- [Creating the Service Author Deliverables](#) — The service authors provide their deliverables.
- [Deploying an On-Demand Broker](#) — The operators upload their releases and write a manifest.



## Getting Started: ODB on a Local Development Environment


This guide describes how to create and manage an on-demand service using PCF Dev and BOSH lite, which are tools that allow you run BOSH and Pivotal Cloud Foundry on a local development machine. This tutorial bases its example service on [Kafka open source messaging](#) and uses the following sample code directories:

- [Kafka example service](#)
- [Kafka example service adapter](#)
- [Kafka example app](#)

## Prerequisites

Before setting up and using ODB on your local development environment, install and configure the following components:

- **BOSH Lite** [↗](#) - *min version v9000.131.0*

 **Note:** For PCF Dev to route requests to the deployments on BOSH Lite ensure you run the script `bin/add-route` in the BOSH Lite repository. You may need to run this again if your networking is reset (e.g. reboot, or connecting to a different network).

- [PCF Dev](#) - [pcfdev-v0.19.1-rc.46](#)
- Once PCF Dev has finished installing, a success message displays.

```
| _ |   |   |   |   |   |   | | | | | | | | | |
| |    ||    | |   | |   ||||  
|_ |    ||   |_|_| |_ |   |_|_|  
| |_|||    || |_| ||| || |_|_|  
|   ||   |_| |_| ||| ||   ||_|  
| | | |_|_| | |   || |_|_|_|  
|_| |_|_|||_| |_|_| |_|_| |_|
```

is now running.

To begin using PCF Dev, please run:

```
cf login -a https://api.local.pcfdev.io --skip-ssl-validation
```

Apps Manager URL: <https://local.pcfdev.io>

Admin user => Email: admin / Password: admin

Make note of the PCF Dev domain. You will need this later on. The default is `local.pcfdev.io`.

## About the BOSH CLI

The BOSH CLI is available in two major versions, v1 and v2. Pivotal recommends that you use the BOSH CLI v2 when possible.

This topic provides examples of using each version of the BOSH CLI. Your PCF installation may affect which version of the BOSH CLI you can use. Consult the table below to determine which version of the CLI is supported for your installation.

PCF Version	BOSH CLI Version
1.10	CLI v1
1.11	CLI v1 or CLI v2 (Pivotal recommends CLI v2)
1.12 and later	CLI v2

## Step 1: Set Up BOSH Lite

1. Target your BOSH Lite installation:

```
$ bosh target
Current target is https://192.168.50.4:25555 (Bosh Lite Director)
```

2. Upload the [BOSH Lite stemcell](#):

```
$ bosh upload stemcell https://bosh.io/d/stemcells/bosh-warden-boshlite-ubuntu-trusty-go_agent?v=3262.2
```

## Step 2: Set Up the Kafka Example Service

1. Clone the [Kafka example service](#) into your workspace:

```
$ git clone https://github.com/pivotal-cf-experimental/kafka-example-service-release.git
```

2. In the `kafka-example-service-release` directory, create and upload the kafka example service:

```
$ cd kafka-example-service-release  
$ bosh create release --name kafka-example-service
```

3. Upload the service to the BOSH director:

```
$ bosh upload release
```

## Step 3: Set Up the Kafka Example Service Adapter

1. Clone the [Kafka example service adapter](#) and run `git submodule update --init` to bring in the adapter's dependencies

```
$ git clone https://github.com/pivotal-cf-experimental/kafka-example-service-adapter-release.git
```

2. Update the service adapter dependencies:

```
$ cd kafka-example-service-adapter-release  
$ git submodule update --init --recursive
```

3. Create the example service adapter:

```
$ bosh create release --name kafka-example-service-adapter
```

4. Upload the example service adapter to the BOSH director:

```
$ bosh upload release
```

## Step 4: Set Up the On-Demand Service Broker

1. Download the [on-demand service broker from PivNet](#)
2. Upload the `on-demand-service-broker` release (replace X.Y.Z with the ODB release version):

```
$ bosh upload release on-demand-service-broker-X.Y.Z.tgz
```

## Step 5: Create a BOSH Deployment

1. Create a new directory in your workspace and a `cloud_config.yml` for the BOSH Lite Director. For example:

```
vm_types:
- name: container
  cloud_properties: {}

networks:
- name: kafka
  type: manual
  subnets:
  - range: 10.244.1.0/24
    gateway: 10.244.1.1
    az: lite
  cloud_properties: {}

disk_types:
- name: ten
  disk_size: 10_000
  cloud_properties: {}

azs:
- name: lite
  cloud_properties: {}

compilation:
  workers: 2
  reuse_compilation_vms: true
  network: kafka
  az: lite
  cloud_properties: {}
```

2. Update the BOSH Lite cloud config using the deployment manifest created in the previous step:

```
$ bosh update cloud-config cloud_config.yml
```

3. Obtain the URL and UUID BOSH Lite director information:

```
$ bosh status
```

This command produces output similar to the following:

```
→ bosh status
Config
  /Users/pivotal/.bosh_config

Director
  Name      Bosh Lite Director
  URL       https://192.168.50.4:25555
  Version   1.3215.0 (00000000)
  User      admin
  UUID      17a45148-1d00-43bc-af28-9882e5a6535a
  CPI       warden_cpi
  dns       disabled
  compiled_package_cache enabled (provider: local)
  snapshots disabled
```

Record the URL and UUID from your output. You will add them to the `deployment_manifest.yml` in the next step.

4. Create a BOSH Lite deployment manifest in a file called `deployment_manifest.yml` using the following as a base.

 Replace `BOSH_LITE_UUID`, `BOSH_LITE_URL` and `PCF_DEV_DOMAIN` with the values obtained in the previous steps.

```
name: kafka-on-demand-broker

director_uuid: <BOSH_LITE_UUID>

releases:
- name: &broker-release on-demand-service-broker
  version: latest
- name: &service-adapter-release kafka-example-service-adapter
  version: latest
- name: &service-release kafka-example-service
  version: latest

stemcells:
- alias: trusty
  os: ubuntu-trusty
```

```

version: <STEMCELL_VERSION>

instance_groups:
- name: broker
  instances: 1
  vm_type: container
  persistent_disk_type: ten
  stemcell: trusty
  azs: [lite]
  networks:
  - name: kafka
  jobs:
  - name: kafka-service-adapter
    release: *service-adapter-release
  - name: admin_tools
    release: *service-release
  - name: broker
    release: *broker-release
  properties:
    port: 8080
    username: broker #or replace with your own
    password: password #or replace with your own
    disable_ssl_cert_verification: true
  bosh:
    url: <BOSH_LITE_URL>
    authentication:
      basic:
        username: admin
        password: admin
  cf:
    url: https://api.<PCF_DEV_DOMAIN>
    authentication:
      url: https://uaa.<PCF_DEV_DOMAIN>
      user_credentials:
        username: admin
        password: admin
  service_adapter:
    path: /var/vcap/packages/odb-service-adapter/bin/service-adapter
  service_deployment:
    releases:
    - name: *service-release
      version: <SERVICE_RELEASE_VERSION>
      jobs: [kafka_server, zookeeper_server]
    stemcell:
      os: ubuntu-trusty
      version: <STEMCELL_VERSION>
  service_catalog:
    id: D94A086D-203D-4966-A6F1-60A9E2300F72
    service_name: kafka-service-with-odb
    service_description: Kafka Service
    bindable: true
    plan_updatable: true
    tags: [kafka]
  plans:
  - name: small
    plan_id: 11789210-D743-4C65-9D38-C80B29F4D9C8
    description: A Kafka deployment with a single instance of each job and persistent disk
    instance_groups:
    - name: kafka_server
      vm_type: container
      instances: 1
      persistent_disk_type: ten
      azs: [lite]
      networks: [kafka]
    - name: zookeeper_server
      vm_type: container
      instances: 1
      persistent_disk_type: ten
      azs: [lite]
      networks: [kafka]
    properties:
      auto_create_topics: true
      default_replication_factor: 1

update:
  canaries: 1
  canary_watch_time: 30000-180000
  update_watch_time: 30000-180000
  max_in_flight: 4

```

5. Change the BOSH deployment to use the deployment manifest created in the previous step:

```
$ bosh deployment deployment_manifest.yml
```

6. Deploy the manifest:

```
$ bosh deploy
```

7. Obtain the IP address of the deployed broker:

```
$ bosh instances
```

This command produces output similar to the following:

```
Acting as client 'admin' on deployment 'kafka-on-demand-broker' on 'Bosh Lite Director'

Director task 147

Task 147 done

+-----+-----+-----+-----+
| Instance | State | AZ | VM Type | IPs |
+-----+-----+-----+-----+
| broker/0 (59231277-d7b8-46bb-8bbb-8154b6bae347)* | running | n/a | container | 10.244.1.2 |
+-----+-----+-----+-----+


(*) Bootstrap node

Instances total: 1
```

Record the IP address of the broker. You will use this in the next step to create a service broker.

## Step 6: Create a Service Broker on PCF Dev

1. Create a service broker on PCF Dev and enable access to its service offering. You will need the broker's credentials set in the deployment manifest and the IP of the broker VM.

 Replace `BROKER_IP` with the value obtained in the previous step.

```
$ cf create-service-broker kafka-broker broker password http://<BROKER_IP>:8080
```

For more details on service brokers see [here](#).

2. Enable access to the broker's service plans:

```
$ cf enable-service-access kafka-service-with-odb
```

3. View the services offered by the broker in the marketplace:

```
$ cf marketplace
```

This command produces output similar to the following:

```
Getting services from marketplace in org pcfdev-org / space pcfdev-space as admin...
OK

service      plans      description
kafka-service-with-odb  small      Kafka Service
p-mysql       512mb, 1gb MySQL databases on demand
p-rabbitmq    standard   RabbitMQ is a robust and scalable high-performance multi-protocol messaging broker.
p-redis       shared-vm  Redis service to provide a key-value store
```

4. Create a service instance using the Kafka on-demand broker.

```
$ cf create-service kafka-service-with-odb small k1
```

## Step 7: Verify Your BOSH Deployment and On-Demand Service

1. Check the status of your service:

```
$ cf service k1
```

Initially, the service status state is: `create in progress`. After the service is created, the status changes to: `create succeeded`.

2. Verify that the on-demand service is provisioned in the BOSH deployment

```
$ bosh deployments
```

The output of this command appears as follows:

```
+-----+-----+-----+-----+
| Name                               | Release(s)                | Stemcell(s)                | Cloud Config |
+-----+-----+-----+-----+
| kafka-on-demand-broker             | kafka-example-service-adapter/0+dev.2 | bosh-warden-boshlite-ubuntu-trusty-go\agent/3262.2 | latest      |
|                                     | on-demand-service-broker/0.2.0+dev.1 |                                     |             |
+-----+-----+-----+-----+
| service-instance\_2715262c-8564-4cd9-b629-0ae99e6aa4b9 | kafka-example-service/0+dev.2 | bosh-warden-boshlite-ubuntu-trusty-go\agent/3262.2 | latest      |
+-----+-----+-----+-----+
```

This example shows that the service instance is provisioned and the service releases are specified in the ODB deployment manifest.

## Step 8: Use Your On-Demand Service

The [Kafka Example App](#) shows how you can use the service instance that you created with `cf create-service`.

1. Clone the [Kafka example app](#) in your workspace:


```
$ git clone https://github.com/pivotal-cf-experimental/kafka-example-app.git
```

2. Push the app.

```
$ cd kafka-example-app
$ cf push --no-start
```

3. Bind the app to your service instance:

```
$ cf bind-service kafka-example-app k1
```

 **Note:** You can use `cf bs` as an alias for `cf bind-service`

4. Start the app:

```
$ cf start kafka-example-app
```

Now the app runs at `https://kafka-example-app.<PCF_DEV_DOMAIN>`, and you can use it to read and write to your on-demand Kafka service instance. For example:

- To write data, run `curl -XPOST http://kafka-example-app.<PCF_DEV_DOMAIN>/queues/my-queue -d SOME-DATA`
- To read data, run `curl http://kafka-example-app.<PCF_DEV_DOMAIN>/queues/my-queue`

## Creating the Service Author Deliverables

### Service Author Requirements

The following deliverables are required from the service authors:

- Service release(s)
- BOSH release(s) to be deployed by the manifest that is generated by the Service Adapter
- Service Adapter BOSH release
- Contains the Service Adapter CLI
- Documentation for the operator to configure plan definitions for the Service Adapter
- Documentation for the operator to backup and restore service instances

For information about what is required of the Operator, see [Responsibilities of the Operator](#).


### Create a Service Release

A service release is a BOSH release that is deployed at instance creation time, once for each service instance, by the on-demand broker (ODB). We have created two examples:

- [Redis](#)
- [Kafka](#)

See the [BOSH docs](#) for help creating a BOSH release. We recommend creating sample manifests that deploy the service release(s), as this will help you write the `generate-manifest` component of the Service Adapter later.

### Service Instance Lifecycle Errands

 **Note:** This feature requires BOSH director v261 or later.

A service release can provide job errands that can be used by ODB during the management of an instance lifecycle. Service instance lifecycle errands may be [configured by the operator](#).

ODB supports the following service instance lifecycle errands:

- `post-deploy` - Runs after the creation or updating of a service instance. See the workflow [here](#).
- `pre-delete` - Runs before the deletion of a service instance. See the workflow [here](#).

A deployment is only considered successful if along with the deployment the lifecycle errand completes successfully.

See an example implementation of a health check post-deploy job in the [example redis release](#).

In the `generate-manifest` command ensure to validate and include any supported errands that are specified in the instance groups array.


### Job links

When generating a manifest, we recommend not using static IPs as this makes network IP management very complex. Instead, we recommend using [BOSH's job links feature](#). There are two types of job links, implicit and explicit. The [example Kafka release](#) uses implicit job links to get the IPs of the brokers and the zookeeper. Details on how to use the links feature are available [here](#).

### Create a Service Adapter

A Service Adapter is an executable invoked by ODB. It is expected to respond to these subcommands:

- `generate-manifest` Generate a BOSH manifest for your service instance deployment and output to stdout as YAML, given information about the:
  - BOSH director (stemcells, release names)
  - service instance (ID, request parameters, plan properties, IAAS resources)
  - previous manifest, if this is an upgrade deployment
- `dashboard-url` Generate an optional URL of a web-based management user interface for the service instance.
- `create-binding` Create (unique, if possible) credentials for the service instance, printing them to stdout as JSON.
- `delete-binding` Invalidate the created credentials, if possible. Some services (e.g. Redis) are single-user, and this endpoint will do nothing.

 **\*\*Note\*\*:** The ODB requires `generate-manifest` to be a pure function when a previous manifest is supplied (updating a deployment scenario): given the same arguments the command should always output the same BOSH manifest.

The parameters, and expected output from these subcommands will be explained in detail below. For each of these subcommands, exit status 0 indicates that the command succeeded exit status 10 indicates not implemented, and any non-zero status indicates failure.

## Handle Errors

If a subcommand fails, the adapter must return a non-zero exit status, and may optionally print to stdout and/or stderr.

When a subcommand exits with an unrecognized exit code anything printed to stdout will be returned to the CF CLI user.


Both the stdout and stderr streams will be printed in the broker log for the operator. For that reason, we recommend not printing the manifest or other sensitive details to stdout/stderr, as the ODB does no validation on this output.

See an example implementation [here](#) .

## Inputs for manifest generation

### Request parameters

The [body](#)  of the provision request from Cloud Controller, including arbitrary parameters from the CLI user.

Service authors can choose to allow Cloud Foundry users to configure service instances with arbitrary parameters. See the PCF docs on [Managing Service Instances with the CLI](#) . Arbitrary parameters can be passed to the service adapter when creating, or updating a service instance. They allow Cloud Foundry users to override the default configuration for a service plan.

Service authors must document the usage of arbitrary parameters for Cloud Foundry users.

For example:


- the [Kafka service adapter](#)  supports the `auto_create_topics` arbitrary parameter to configure auto-creation of topics on the cluster.

### Previous Manifest Properties

Service authors can choose to migrate certain properties for the service from the previous manifest when updating a service instance. If the previous manifest is ignored then any properties configured using arbitrary parameters will not be migrated when a service instance is updated.

Service authors must document the migration of previous manifest properties for operators.

For example:

- the [Kafka service adapter](#)  supports migration of the `auto_create_topics` previous plan property to configure auto-creation of topics on the cluster.

### Service Plan Properties

Service authors can choose to support certain properties for the service in the adapter code. These properties are service-specific traits used to customize



the service. They do not necessarily map to jobs one to one; a plan property may affect multiple jobs in the deployment. Plan properties are a mechanism for the operator to define different plans.

Service authors must document the usage of plan properties for the operator.

For example:

- the [Redis service adapter](#) supports the `persistence` property which can be used to attach a disk to the vm.
- the [Kafka service adapter](#) supports the `auto_create_topics` property to enable auto-creation of topics on the cluster.

## Order of Precedence

Note, we recommend service authors use the following order of precedence in their service adapters when generating manifests:

1. arbitrary parameters
2. previous manifest properties
3. plan properties

For example, see `auto_create_topics` in the [example Kafka service adapter](#).

## Service Adapter Interface

A service adapter is expected to be implemented as a binary with the interface

```
service-adapter [subcommand] [params ...]
```

where the subcommand can be generate-manifest, create-binding, delete-binding


Examples are provided for [Redis](#) and [Kafka](#). Note that these Golang examples use the SDK to help with cross-cutting concerns such as unmarshalling the JSON command line parameters. For example, see the use of `HandleCommandLineInvocation` in the [redis-adapter](#).

## Subcommands

### generate-manifest

```
service-adapter generate-manifest [service-deployment-JSON] [plan-JSON] [request-params-JSON] [previous-manifest-YAML] [previous-plan-JSON]
```

The generate-manifest subcommand takes in 5 arguments and returns a BOSH deployment manifest YAML.

 **Note:** The ODB requires generate-manifest to be a pure function when a previous manifest is supplied (updating a deployment scenario): given the same arguments the command should always output the same BOSH manifest.

### Output

The following table describes the supported exit codes and output for the `generate-manifest` subcommand:

#### Supported Exit Codes for generate-manifest

exit code	Description	Output
0	success	Stdout: BOSH manifest YAML

Exit code	Description	Output
anything else	failure	Stdout: optional error message for CF CLI users Stderr: error message for operator ODB will log both stdout and stderr

## Parameters

### service-deployment-JSON

Provides information regarding the BOSH director

field	Type	Description
deployment_name	string	name of the deployment on the director, in the format <code>service-instance_\${guid}</code>
releases	array of releases	list of service releases configured for the deployment by the operator
release.name	string	name of the release on the director
release.version	string	version of the release
release.jobs	array of strings	list of jobs required from the release
stemcell	map	the stemcell available on the director
stemcell.stemcell_os	string	stemcell OS available on the director
stemcell.stemcell_version	string	stemcell version available on the director

For example

```
{
  "deployment_name": "service-instance_${GUID}",
  "releases": [{
    "name": "kafka",
    "version": "dev.42",
    "jobs": [
      "kafka_node",
      "zookeeper"
    ]
  }],
  "stemcell": {
    "stemcell_os": "BeOS",
    "stemcell_version": "2"
  }
}
```

ODB only supports injecting one stemcell into each service deployment (different instance groups cannot have different stemcells).

ODB only supports using exact release and stemcell versions. The use of `latest` and floating stemcells are not supported.

Your Service Adapter should be opinionated about which jobs it requires to generate its manifest. For example, the Kafka example requires `kafka_node` and `zookeeper`. It should not be opinionated about the mapping of BOSH release to job. The jobs can all be provided by one release, or across many. The SDK provides the helper function [GenerateInstanceGroupsWithNoProperties](#) for generating instance groups without any properties. The Kafka example service adapter [uses this helper function](#) and [invokes it](#) to map the service releases parameter to the BOSH manifest `releases` and `instance_groups` sections.

You should provide documentation about which jobs are required by your Service Adapter, and which BOSH releases operators should get these jobs from.

### plan-JSON

Plan for which the manifest is supposed to be generated

### plan-JSON schema

field	Type	Description
-------	------	-------------

field	type	description
instance_groups	array of instance groups	instance groups configured for the plan
instance_group.name	string	name of the instance group
instance_group.vm_type	string	the vm_type configured for the instance group, matches one in the cloud config on the director
instance_group.vm_extensions	array of strings	Optional, the vm_extensions configured for the instance group, must be present in the cloud config on the director
instance_group.persistent_disk_type	string	Optional, the persistent_disk_type configured for the instance group, matches one in the cloud config on the director
instance_group.networks	array of strings	the networks the instance group is supposed to be in
instance_group.instances	int	number of instances for the instance group
instance_group.lifecycle	string	Optional, specifies the kind of workload the instance group represents. Valid values are <code>service</code> and <code>errand</code> ; defaults to <code>service</code>
instance_group.azs	array of strings	a list of availability zones that the instance groups should be striped across
properties	map	properties which the operator has configured for deployments of the current plan
update	map	update block which the operator has configured for deployments of the current plan
update.canaries	int	plan-specific number of canary instances
update.max_in_flight	int	plan-specific maximum number of non-canary instances to update in parallel
update.canary_watch_time	string	plan-specific time in milliseconds that the BOSH Director sleeps before checking whether the canary instances are healthy
update.update_watch_time	string	plan-specific time in milliseconds that the BOSH Director sleeps before checking whether the non-canary instances are healthy
update.serial	boolean	Optional, plan-specific flag to deploy instance groups sequentially ( <code>true</code> ), or in parallel ( <code>false</code> ); defaults to <code>true</code>

For example

```
{
  "instance_groups": [
    {
      "name": "example-server",
      "vm_type": "small",
      "vm_extensions": ["some", "extensions"],
      "persistent_disk_type": "ten",
      "networks": [
        "example-network"
      ],
      "azs": [
        "example-az"
      ],
      "instances": 1
    },
    {
      "name": "example-migrations",
      "vm_type": "small",
      "persistent_disk_type": "ten",
      "networks": [
        "example-network"
      ],
      "instances": 1,
      "lifecycle": "errand"
    }
  ],
  "properties": {
    "example": "property"
  },
  "update": {
    "canaries": 1,
    "max_in_flight": 2,
    "canary_watch_time": "1000-30000",
    "update_watch_time": "1000-30000",
    "serial": true
  }
}
```

Plans are composed by the operator and consist of resource mappings, properties and an optional update block:

- **Resource Mappings**

The `instance_groups` section of the plan JSON. This maps service deployment instance groups (defined by the service author) to resources (defined by the operator). The service developers should document the list of instance group names required for their deployment (e.g. “redis-server”) and any constraints they recommend on resources (e.g. operator must add a persistent disk if persistence property is enabled). These constraints can of course be enforced in code. The `instance_groups` section also contains a field for `lifecycle`, which can be set by the operator. The service adapter will add a lifecycle field to the instance group within the BOSH manifest when specified.

- **Properties**

Properties are service-specific parameters chosen by the service author. The Redis example exposes a property `persistence`, which takes a boolean value and toggles disk persistence for Redis. These should be documented by the service developers for the operator.

- **Update Block (optional)**

This block defines a plan-specific configuration for BOSH’s update instance operation. Although the ODB considers this block optional, the service adapter must output an update block in every manifest it generates. Some ways to achieve that are:

1. *(Recommended)* Define a default update block for all plans, which is used when a plan-specific update block is not provided by the operator
2. Hard code an update block for all plans in the service adapter
3. Make the update block mandatory, so that operators must provide an update block for every plan in the service catalogue section of the ODB manifest

## request-params-JSON

This is a JSON object that holds the entire body of the [service provision](#) or [service update](#) request sent by the Cloud Controller to the service broker. The request parameters JSON will be `null` for upgrades.

The field `parameters` contains arbitrary key-value pairs which were passed by the application developer as a `cf` CLI parameter when creating, or updating the service instance.

Note: when updating an existing service instance, any arbitrary parameters passed on a previous create or update will not be passed again. Therefore, for arbitrary parameters to stay the same across multiple deployments they must be retrieved from the previous manifest.

## previous-manifest-YAML

The previous manifest as YAML. The previous manifest is nil if this is a new deployment. The format of the manifest should match the [BOSH v2 manifest](#).

It is up to the service author to perform any necessary service-specific migration logic here, if previous manifest is non-nil.

Another use-case of the previous manifest is for the migration of deployment properties which need to stay the same across multiple deployments of a manifest. For example in the Redis example, we [generate a password](#) when we do a new deployment. But when the previous deployment manifest is provided, we copy the password over from [the previous deployment](#), as generating a new password for existing deployments will break existing bindings.

For example see the [example Redis service adapter](#).

## previous-plan-JSON

The previous plan as JSON. The previous plan is nil if this is a new deployment. The format of the plan should match [plan schema](#). The previous plan can be used for complex plan migration logic, for example the [kafka service adapter](#), rejects a plan migration if the new plan reduces the number of instances, to prevent data loss.

## dashboard-url

```
service-adapter dashboard-url [instance-ID] [plan-JSON] [manifest-YAML]
```

The `dashboard-url` subcommand takes in 3 arguments and returns a JSON with the `dashboard_url`. The dashboard URL is optional. If no dashboard URL is relevant to the service, the subcommand should exit with code 10. Provisioning will be successful without the dashboard URL.

## Output

If the `dashboard-url` command generates a url successfully, it should exit with 0 and return a dashboard URL JSON with the following structure:

field	Type	Description
dashboard_url	string	dashboard url returned to the cf user

```
{
  "dashboard_url": "https://someurl.example.com"
}
```

## Supported exit codes for dashboard-url

exit code	Description	Output
0	success	Stdout: dashboard URL JSON
10	not implemented	
anything else	failure	Stdout: optional error message for CF CLI users Stderr: error message for operator ODB will log both stdout and stderr

## instance-ID

Provided by the cloud controller which uniquely identifies the service-instance.

## plan-JSON

Current plan for the service instance as JSON. The structure should be the same as the [plan given in the generate manifest](#)

## manifest-YAML

The current manifest as YAML. The format of the manifest should match the [BOSH] v2 manifest](<https://bosh.io/docs/manifest-v2.html>)

## create-binding

```
service-adapter create-binding [binding-ID] [bosh-VMs-JSON] [manifest-YAML] [request-params-JSON]
```

Binding credentials for a service instance should share a namespace, and should be unique if possible. E.g. for MySQL, two bindings could include a different username/password pairs, but share the same MySQL database tables and data. The first step is to determine which credentials are best to supply in the context of your service. We recommend that users can be identified statelessly from the binding ID, and the simplest way to do this is to name the user after the binding ID.

## Output

If the `create-binding` command is successful, it should return an exit code of 0 and print a [service broker API binding JSON response](#) on stdout. An example response is shown below. If the command failed, it should return any non-zero exit code, see the [supported exit code table](#) for details of supported failure cases. Stdout and stderr from the command will be logged by the ODB.

Example success response to `create-binding` :

```
{
  "credentials": {
    "username": "user1",
    "password": "reallysecret"
  },
  "syslog_drain_url": "optional: for syslog drain services only",
  "route_service_url": "optional: for route services only"
}
```

## Supported exit codes for binding

exit code	Description	Output
0	success	Stdout: binding credentials JSON
10	subcommand not implemented	
42	app_guid not provided in the binding request body	Stderr: error message for operator ODB will log both stdout and stderr
49	binding already exists	Stderr: error message for operator ODB will log both stdout and stderr
anything else	failure	Stdout: optional error message for CF CLI users Stderr: error message for operator ODB will log both stdout and stderr

## Parameters

### binding-ID

The binding-ID generated by the Cloud Controller.

### bosh-VMs-JSON

A map of instance group name to an array of IPs provisioned for that instance group.

For example

```
{
  "mysql_node": ["192.0.2.1", "192.0.2.2", "192.0.2.3"],
  "management_box": ["192.0.2.4"]
}
```

This can be used to connect to the instance deployment if required, to create a service specific binding. In the example above, the Service Adapter may connect to MySQL as the admin and create a user. As part of the binding, the `mysql_node` IPs would be returned, but maybe not the `management_box`.

### manifest-YAML

The current manifest as YAML. This is used to extract information about the deployment that is necessary for the binding (e.g. admin credentials with which to create users). The format of the manifest should match the [BOSH v2 manifest](#).

### request-params-JSON

This is a JSON object that holds the entire body of the [service binding](#) request sent by the Cloud Controller to the service broker.

The field `parameters` contains arbitrary key-value pairs which were passed by the application developer as a `cf` CLI parameter when creating, or updating the service instance.

## Credentials for Bindings

We have identified three approaches to credentials for a service binding.

### Static Credentials

In this case, the same credentials are used for all bindings. One option is to define these credentials in the service instance manifest.

This scenario makes sense for services that use the same credentials for all bindings, such as Redis. For example:

```
properties:
  redis:
    password: <same-for-all-bindings>
```

### Credentials Unique to Each Binding

In this case, when the adapter `generate-manifest` subcommand is invoked, it generates random admin credentials and returns them as part of the service instance manifest. When the `create-binding` subcommand is invoked, the adapter can use the admin credentials from the manifest to create unique credentials for the binding. Subsequent `create-binding` s create new credentials.

This option makes sense for services whose binding creation resembles user creation, such as MySQL or RabbitMQ. For example, in MySQL the admin user can be used to create a new user and database for the binding:

```
properties:
  admin_password: <use-to-create-credentials>
```

### Using an Agent

In this case, the author defines an agent responsible for handling creation of credentials unique to each binding. The agent must be added as a BOSH release in the service manifest. Moreover, the service and agent jobs should be co-located in the same instance group.

This option is useful for services where the adapter cannot or prefers not to directly call out to the service instance, and instead delegates responsibility for setting up new credentials to an agent.

For example:

```
releases:
  - name: service-release
    version: 1.5.7
  - name: credentials-agent-release
    version: 4.2.0

instance_groups:
  - name: service-group
    jobs:
      - name: service-job
        release: service-release
      - name: credentials-agent-job
        release: credentials-agent-release
```

---

## delete-binding

```
service-adapter delete-binding [binding-ID] [bosh-VMs-JSON] [manifest-YAML] [request-params-JSON]
```

This should invalidate the credentials that were generated by `create-binding` if possible. E.g. for MySQL, it would delete the binding user.

### Output

The following table describes the supported exit codes and output for the `delete-binding` subcommand:

## Supported exit codes for delete-binding

exit code	Description	Output
0	success	No output is required
10	not implemented	
41	binding does not exist	Stderr: error message for operator ODB will log both stdout and stderr
anything else	failure	Stdout: optional error message for CF CLI users Stderr: error message for operator ODB will log both stdout and stderr

## Parameters

### binding-ID

The binding to be deleted.

### bosh-VMs-JSON

A map of instance group name to an array of IPs provisioned for that instance group.

For example

For example

```
{
  "my-instance-group": ["192.0.2.1", "192.0.2.2", "192.0.2.3"]
}
```

This can be used to connect to the actual VMs if required, to delete a service specific binding. For example delete a user in MySQL.

### manifest-YAML

The current manifest as YAML. This is used to extract information about the deployment that is necessary for the binding (e.g. credentials). The format of the manifest should match the [BOSH v2 manifest](#).

For example see the [kafka delete binding](#).

## Request Params JSON

This is a JSON object that holds the entire body of the [service unbinding](#) request sent by the Cloud Controller to the service broker.

The field `parameters` contains arbitrary key-value pairs which were passed by the application developer as a `cf` CLI parameter when creating, or updating the service instance.

## Packaging

This topic describes workflows for setting up and maintaining of a service instance. The diagrams show which tasks are undertaken by the ODB and which require interaction with the Service Adapter.

The adapter should be packaged as a BOSH release, which should be co-located with the ODB release in a BOSH manifest by the operator. This is only done in order to place the adapter executable on the same VM as the ODB server, therefore the adapter BOSH job's `monit` file should probably have no processes defined.

Example service adapter releases:



- [Kafka](#)
- [Redis](#)

## Golang SDK

We have published a [SDK](#) for teams writing their service adapters in Golang. It encapsulates the command line invocation handling, parameter parsing, response serialization and error handling so the adapter authors can focus on the service-specific logic in the adapter.

You should use the same version of the SDK as your ODB release. For example if you are using v0.8.0 of the ODB BOSH release you should checkout the v0.8.0 tag of the SDK.

For the generated BOSH manifest the SDK supports properties in two levels: manifest (global) and job level. Global properties are [deprecated in BOSH](#), in favour of job level properties and job links. As an example, refer to the [Kafka example service adapter property generation](#).

## Usage

Get the SDK:

```
go get github.com/pivotal-cf/on-demand-services-sdk
```

In the main function for the service adapter, call the `HandleCommandLineInvocation` function:

```
package main

import (
    "log"
    "os"

    "github.com/bar-org/foo-service-adapter/adapter"
    "github.com/pivotal-cf/on-demand-services-sdk/serviceadapter"
)

func main() {
    logger := log.New(os.Stderr, "[foo-service-adapter] ", log.LstdFlags)
    manifestGenerator := adapter.ManifestGenerator{}
    binder := adapter.Binder{}
    dashboardUrlGenerator := adapter.DashboardUrlGenerator{}
    serviceadapter.HandleCommandLineInvocation(os.Args, manifestGenerator, binder, dashboardUrlGenerator)
}
```

## Interfaces

The `HandleCommandLineInvocation` function accepts structs that implement these interfaces:

```
type ManifestGenerator interface {
    GenerateManifest(serviceDeployment ServiceDeployment, plan Plan, requestParams RequestParameters, previousManifest *bosh.BoshManifest, previousPlan *Plan) (bosh.BoshManifest, error)
}

type Binder interface {
    CreateBinding(bindingID string, deploymentTopology bosh.BoshVMs, manifest bosh.BoshManifest, requestParams RequestParameters) (Binding, error)
    DeleteBinding(bindingID string, deploymentTopology bosh.BoshVMs, manifest bosh.BoshManifest, requestParams RequestParameters) error
}

type DashboardUrlGenerator interface {
    DashboardUrl(instanceID string, plan Plan, manifest bosh.BoshManifest) (DashboardUrl, error)
}
```

## Helpers

The helper function `GenerateInstanceGroupsWithNoProperties` can be used to generate the instance groups for the BOSH manifest from the arguments passed to the adapter. One of the inputs for this function is the mapping of instance groups to jobs for the deployment (`deploymentInstanceGroupsToJobs`). This mapping must be provided by the service author. This function will not address job level properties for the generated instance groups; these properties must also

be provided by the service author. For an example implementation see the [job mapping in the Kafka example adapter](#).

## Error handling

Any error returned by the interface functions is considered to be for the Cloud Foundry CLI user and will accordingly be printed to stdout.

The adapter code is responsible for performing any error logging to stderr that the authors think is relevant for the operator logs.

There are three specialised errors for the `CreateBinding` function, which allow the adapter to exit with the appropriate code:

```
serviceadapter.NewBindingAlreadyExistsError()  
serviceadapter.NewBindingNotFoundError()  
serviceadapter.NewAppGuidNotProvidedError()
```

For more complete code examples please take a look at the [kafka adapter](#) or the [redis adapter](#).

## Operating an On-Demand Broker

### Operator Responsibilities

The operator is responsible for performing the following:

- Request appropriate networking rules for on-demand service tiles.
- Configure the BOSH director
- Upload the required releases for the broker deployment and service instance deployments.
- Write a broker manifest
  - See [v2-style manifest docs](#) if unfamiliar with writing BOSH v2 manifests
  - Core broker configuration
  - Service catalog and plan composition
- Manage brokers
- Documentation for the operator

For a list of deliverables provided by the Service Author, see [Required Deliverables](#).

For an example manifest for a Redis service, see [redis-example-service-adapter-release](#).

For an example manifest for a Kafka service, see [kafka-example-service-adapter-release](#).

### About the BOSH CLI

The BOSH CLI is available in two major versions, v1 and v2. Pivotal recommends that you use the BOSH CLI v2 when possible.

This topic provides examples of using each version of the BOSH CLI. Your PCF installation may affect which version of the BOSH CLI you can use. Consult the table below to determine which version of the CLI is supported for your installation.

PCF Version	BOSH CLI Version
1.10	CLI v1
1.11	CLI v1 or CLI v2 (Pivotal recommends CLI v2)
1.12 and later	CLI v2

### Set Up Your BOSH Director

Dependencies for the On-Demand Broker:

- BOSH director v257 or later (PCF 1.8) **Note:** does not support BOSH Windows
- Cloud Foundry v238 or later (PCF 1.8)



**Note:** [Service instance lifecycle errands](#) require BOSH director v261 (PCF 1.10) or later.

### SSL Certificates

If ODB is configured to communicate with BOSH on the director's public IP you may be using a self-signed certificate unless you have a domain for your BOSH director. ODB does not ignore TLS certificate validation errors by default (as expected). You have two options to configure certificate-based authentication between the BOSH director and the ODB:

1. Add the BOSH director's root certificate to ODB's trusted pool in the ODB manifest:

```
bosh:
  root_ca_cert: <root-ca-cert>
```

2. Use BOSH's `trusted_certs` feature to add a self-signed CA certificate to each VM BOSH deploys. For more details on how to generate and use self-signed certificates for BOSH director and UAA, see [Director SSL Certificate Configuration](#).

You can also configure a separate root CA certificate that is used when ODB communicates with the Cloud Foundry API (Cloud Controller). This is done in a similar way to above. Please see [manifest snippets below](#) for details.

## BOSH Teams

BOSH has a teams feature that allows you to further control how BOSH operations are available to different clients. We strongly recommend using it to ensure that your on-demand service broker client can only modify deployments it created. For example, if you [use uaac](#) to create the client like this:

```
uaac client add <client-id> \  
--secret <client-secret> \  
--authorized_grant_types "refresh_token password client_credentials" \  
--authorities "bosh.teams.<team-name>.admin"
```

Then when you [configure the broker's BOSH authentication](#), you can use this client ID and secret. The broker will then only be able to perform BOSH operations on deployments it has created itself.

For more details on how to set up and use BOSH teams, see [Director teams and permissions configuration](#).

For more details on securing how ODB uses BOSH, see [Security](#).

## Cloud Controller

ODB used the Cloud Controller as a source of truth about service offerings, plans, and instances. To reach Cloud Controller, ODB needs to be configured with credentials. These can be either client or user credentials:

- Client credentials: as of Cloud Foundry v238, the UAA client must have authority `cloud_controller.admin`.
- User credentials: a Cloud Foundry admin user, i.e. a member of the `scim.read` and `cloud_controller.admin` groups as a minimum.

Detailed broker configuration is covered [below](#).

## Upload Required Releases

Upload the following releases to the BOSH director:

- on-demand-service-broker
- your service adapter
- your service release(s)

## Write a Broker Manifest

### Core Broker Configuration

Your manifest should contain one non-errand instance group, that co-locates both:

- the `broker` job from on-demand-service-broker
- your service adapter job from your service adapter release

The broker is stateless and does not need a persistent disk. The VM type can be quite small: a single CPU and 1 GB of memory should be sufficient in most cases.

An example snippet is shown below:

```
instance_groups:
- name: broker # this can be anything
  instances: 1
  vm_type: <vm type>
  stemcell: <stemcell>
  networks:
  - name: <network>
  jobs:
  - name: <service adapter job name>
    release: <service adapter release>
  - name: broker
    release: on-demand-service-broker
  properties:
    # choose a port and basic auth credentials for the broker
    port: <broker port>
    username: <broker username>
    password: <broker password>
    disable_ssl_cert_verification: <true|false> # optional, defaults to false. This should NOT be used in production
  cf:
    url: <CF API URL>
    root_ca_cert: <ca cert for cloud controller> # optional, see SSL certificates
    authentication: # either client_credentials or user_credentials, not both as shown
    url: <CF UAA URL>
    client_credentials:
      client_id: <UAA client id with cloud_controller.admin authority and client_credentials in the authorized_grant_type>
      secret: <UAA client secret>
    user_credentials:
      username: <CF admin username in the cloud_controller.admin and scim.read groups>
      password: <CF admin password>
  bosh:
    url: <director url>
    root_ca_cert: <ca cert for bosh director and associated UAA> # optional, see SSL certificates
    authentication: # either basic or uaa, not both as shown
    basic:
      username: <bosh username>
      password: <bosh password>
    uaa:
      url: <BOSH UAA URL> # often on the same host as the director, on a different port
      client_id: <bosh client id>
      client_secret: <bosh client secret>
  service_adapter:
    path: <path to service adapter binary> # optional, provided by the Service Author. Defaults to /var/vcap/packages/odb-service-adapter/bin/service-adapter

# There are more broker properties that are discussed below
```

This snippet is using the BOSH v2 syntax, and making use of global cloud config and job-level properties.

Please note that the `disable_ssl_cert_verification` option is dangerous and **should not be used in production**.

## Service Catalog and Plan Composition

The operator must:

1. Supply each release job specified by the Service Author exactly once. You can include releases that provide many jobs, as long as each required job is provided by exactly one release.
2. Supply one stemcell that is used on each VM in the service deployments. ODB does not currently support service instance deployments that use a different stemcell for different instance groups.
3. Use exact versions for releases and stemcells. The use of `latest` and floating stemcells are not supported.
4. Create Cloud Foundry [service metadata](#) in the catalog for the service offering. This metadata will be aggregated in the Cloud Foundry marketplace and displayed in Apps Manager and the `cf` CLI.
5. Compose plans. In ODB, service authors do not define plans but instead expose plan properties. The operator's role is to compose combinations of these properties, along with IAAS resources and catalog metadata into as many plans as they like.
  - a. Create Cloud Foundry [service plan metadata](#) in the service catalog for each plan.
  - b. Provide resource mapping for each instance group specified by the Service Author for each plan. The resource values must correspond to valid resource definitions in the BOSH director's global cloud config. In some cases Service Authors will recommend resource configuration: e.g. in single-node Redis deployments, an instance count greater than 1 does not make sense. Here the operator can configure the deployment to span multiple availability zones, by using the [BOSH multi-az feature](#). For example the [kafka multi az plan](#). In some cases, service authors will provide errands for the service release. You can add an instance group of type errand by setting the lifecycle field. For example the [smoke\\_tests for the kafka deployment](#).

- c. Provide values for plan properties. Plan properties are key-value pairs defined by the Service Author. Some examples include a boolean to enable disk persistence for Redis, and a list of strings representing RabbitMQ plugins to load. The Service Author should document whether these properties are mandatory or optional, whether the use of one property precludes the use of another, and whether certain properties affect recommended instance group to resource mappings.

Properties can also be specified at the service offering level, where they will be applied to every plan. If there is a conflict between global and plan-level properties, the plan properties will take precedence. 1. Provide an (optional) update block for each plan. You may require plan-specific configuration for BOSH's update instance operation. The ODB passes the plan-specific update block to the service adapter. Plan-specific update blocks should have the same structure as [the update block in a BOSH manifest](#) [↗](#). The Service Author can define a default update block to be used when a plan-specific update block is not provided, and whether the service adapter supports their configuration in the manifest.

Add the snippet below to your broker job properties section:

```

service_deployment:
  releases:
    - name: <service-release>
      version: <service-release-version> # Exact release version
      jobs: [<release-jobs-needed-for-deployment-and-lifecycle-errands>] # Service Author will specify list of jobs required
  stemcell: # every instance group in the service deployment has the same stemcell
  os: <service-stemcell>
  version: <service-stemcell-version> # Exact stemcell version
service_catalog:
  id: <CF marketplace ID>
  service_name: <CF marketplace service offering name>
  service_description: <CF marketplace description>
  bindable: <truelfalse>
  plan_updatable: <truelfalse> # optional
  tags: [<tags>] # optional
  requires: [<required permissions>] # optional
  dashboard_client: # optional
  id: <dashboard OAuth client ID>
  secret: <dashboard OAuth client secret>
  redirect_uri: <dashboard OAuth redirect URI>
  metadata: # optional
  display_name: <display name>
  image_url: <image url>
  long_description: <long description>
  provider_display_name: <provider display name>
  documentation_url: <documentation url>
  support_url: <support url>
  global_properties: {} # optional, applied to every plan.
  global_quotas: # optional
  service_instance_limit: <instance limit> # the maximum number of service instances across all plans
plans:
  - name: <CF marketplace plan name>
    plan_id: <CF marketplace plan id>
    description: <CF marketplace description>
    cf_service_access: <enable/disable/manual> # optional, enable by default.
    bindable: <truelfalse> # optional. If specified, this takes precedence over the bindable attribute of the service
    metadata: # optional
    display_name: <display name>
    bullets: [<bullet1>, <bullet2>]
    costs:
      - amount:
          <currency code (string)>: <currency amount (float)>
          unit: <frequency of cost>
    quotas: # optional
    service_instance_limit: <instance limit> # the maximum number of service instances for this plan
  instance_groups: # resource mapping for the instance groups defined by the Service Author
  - name: <service author provided instance group name>
    vm_type: <vm type>
    vm_extensions: [<vm extensions>] # optional
    instances: <instance count>
    networks: [<network>]
    azs: [<az>]
    persistent_disk_type: <disk> # optional
  - name: <service author provided lifecycle errand name> # optional
    lifecycle: errand
    vm_type: <vm type>
    instances: <instance count>
    networks: [<network>]
    azs: [<az>]
  properties: {} # valid property key-value pairs are defined by the Service Author
  update: # optional
  canaries: 1 # required
  max_in_flight: 2 # required
  canary_watch_time: 1000-30000 # required
  update_watch_time: 1000-30000 # required
  serial: true # optional
  lifecycle_errands: #optional
  post_deploy: <errand name> #optional
  pre_delete: <errand name> #optional

```

## Route Registration

You can optionally colocate the `route_registrar` job from the [routing release](#) with the on-demand-service-broker, in order to:

1. load balance multiple instances of ODB using Cloud Foundry's router
2. access ODB from the public internet

To do this, upload the release to your BOSH director and [configure the job properties](#), replacing the version in that docs URL query string as appropriate.

Remember to set the `broker_uri` property in the [register-broker errand](#) if you configure a route.

## Service Instance Quotas

ODB offers global and plan level service quotas to set service instance limits.

Plan quotas restrict the number of service instances for a given plan, while the global limit restricts the number of service instances across all plans.

When creating a service instance, ODB will check the global service instance limit. If it has not been reached, it will check the plan service instance limit.

Note: These limits do not include orphans. See [listing](#) and [deleting](#) orphans.

## Broker Metrics

The ODB bosh release contains a metrics job, that can be used to emit metrics when colocated with [service metrics](#). You must include the [loggregator release](#) in order to do this.

Add the following jobs to the broker instance group:

```
- name: service-metrics
  release: service-metrics
  properties:
    service_metrics:
      execution_interval_seconds: <interval between successive metrics collections>
      origin: <origin tag for metrics>
      monit_dependencies: [broker] # hardcode this
- name: metron_agent
  release: loggregator
  properties:
    metron_agent:
      deployment: <deployment tag for metrics>
    metron_endpoint:
      shared_secret: <metron secret>
    loggregator:
      etcd:
        machines: [<CF etcd IPs>]
    loggregator_endpoint:
      shared_secret: <loggregator secret>
- name: service-metrics-adapter
  release: <ODB release>
```

An example of how the service metrics can be configured for an on-demand-broker deployment can be seen in the [kafka-example-service-adapter-release](#) manifest.

We have tested this example configuration with loggregator v58 and service-metrics v1.5.0.

Please see the [service metrics docs](#) for more details on service metrics.


## Broker Startup Checks

The following startup checks occur:

- Verify that the CF and BOSH versions satisfy the minimum requirement. **Note:** If your service offering includes lifecycle errands, the minimum requirement for BOSH will be higher. See [Set Up Your BOSH Director](#). If your system does not meet minimum requirements, you will see an insufficient version error. For example: `CF API error: Cloud Foundry API version is insufficient, ODB requires CF v238+.`
- Verify that, for the service offering, no plan IDs have changed for plans that have existing service instances. If there are instances, you will see following error: `You cannot change the plan_id of a plan that has existing service instances.`

## Service Instance Lifecycle Errands



 **Note:** This feature requires BOSH director v261 or later.

Service Instance lifecycle errands allow additional short lived jobs to be run as part of service instance deployment. A deployment is only considered successful if the lifecycle errand successfully exits.

The service adapter must offer this errand as part of the service instance deployment.

ODB supports the following lifecycle errands:

- `post_deploy` - Runs after the creation or updating of a service instance. An example use case is running a health check to ensure the service instance is functioning. See the workflow [here](#)
- `pre_delete` - Runs before the deletion of a service instance. An example use case is cleaning up data prior to a service shutdown. See the workflow [here](#)

Service Instance lifecycle errands are configured on a per-plan basis. To enable lifecycle errands, the errand job must be:

- Added to the service instance deployment.
- Added to the plan's instance groups.
- Set in the plan's lifecycle errands configuration.

An example manifest snippet configuring lifecycle errands for a plan:

```
service_deployment:
  releases:
    - name: <service-release>
      version: <service-release-version>
  jobs:
    - <service_release_job>
    - <post_deploy_errand_job>
    - <pre_delete_errand_job>
  service_catalog:
    plans:
      - name: <CF marketplace plan name>
        lifecycle_errands:
          post_deploy: <post_deploy_errand_job>
          pre_delete: <pre_delete_errand_job>
        instance_groups:
          - name: <service_release_job>
            ...
          - name: <post_deploy_errand_job>
            lifecycle: errand
            vm_type: <vm type>
            instances: <instance count>
            networks: [<network>]
            azs: [<az>]
          - name: <pre_delete_errand_job>
            lifecycle: errand
            vm_type: <vm type>
            instances: <instance count>
            networks: [<network>]
            azs: [<az>]
```

Please note that changing a plan's lifecycle errands configuration while an existing deployment is in progress is not supported. Lifecycle errands will not be run.

## Broker Management

Management tasks on the broker are performed with BOSH errands.

### Register Broker

This errand registers the broker with Cloud Foundry and enables access to plans in the service catalog. The errand should be run whenever the broker is re-deployed with new catalog metadata to update the Cloud Foundry catalog.

Please note that if the `broker_uri` property is set, then you should also register a route for your broker with Cloud Foundry. See [Route registration](#) section for more details.

When `enable_service_access: false` is set, the errand will not change service access for any plan.

Individual plans can be enabled via the optional `cf_service_access` property. This property accepts three values: `enable`, `disable`, `manual`.

- `cf_service_access: enable`: register-broker errand will enable access for that plan
- `cf_service_access: disable`: register broker errand will disable access for that plan
- `cf_service_access: manual`: register-broker errand will perform no action

If the `cf_service_access` property is not set at all, the register-broker errand will enable access for that plan.

Plans with disabled service access will not be visible to non-admin Cloud Foundry users (including Org Managers and Space Managers). Admin Cloud Foundry users can see all plans including those with disabled service access.

Add the following instance group to your manifest:

```
- name: register-broker
  lifecycle: errand
  instances: 1
  jobs:
    - name: register-broker
      release: <odb-release-name>
      properties:
        broker_name: <broker-name>
        broker_uri: <broker URI, only required when a route has been registered> # optional
        disable_ssl_cert_verification: <truelfalse> # defaults to false
        enable_service_access: <truelfalse> # defaults to true
      cf:
        api_url: <cf-api-url>
        admin_username: <cf-api-admin-username>
        admin_password: <cf-api-admin-password>
      vm_type: <vm-type>
      stemcell: <stemcell>
      networks: [{name: <network>}]
      azs: [<az>]
```

Run the errand with `bosh run errand register-broker`.

## Delete All Service Instances and Deregister Broker

This errand performs a similar operation to the errand [delete-all-service-instances](#) and [deregister-broker](#). In addition, it will also disable service access to the service offering before deleting all the instances, and then deregister the broker after all instances have been successfully deleted. We disable service access to ensure that new instances cannot be provisioned during the lifetime of the errand.

The errand will do the following operations:

1. Disable service access to the service offering for all orgs and spaces.
2. Unbind all applications from the service instances
3. Delete all service instances sequentially
4. Deregister the broker from Cloud Foundry

**This errand should only be used with extreme caution when you want to totally destroy all of the on-demand service instances and deregister the broker from the Cloud Foundry.**

Add the following instance group to your manifest:

```
- name: delete-all-service-instances-and-deregister-broker
lifecycle: errand
instances: 1
jobs:
  - name: delete-all-service-instances-and-deregister-broker
    release: <odb-release-name>
    properties:
      broker_name: <broker-name>
      polling_interval: <interval in seconds when waiting for service instance to be deleted> # defaults to 60
      polling_initial_offset: <offset in seconds before starting to poll Cloud Foundry to check if the instance has been deleted> # defaults to 5

vm_type: <vm-type>
stemcell: <stemcell>
networks: [{name: <network>}]
azs: [<az>]
```



#### Notes:

The `polling_interval` default is set to 60 seconds because the Cloud Controller itself polls the on-demand broker every 60 seconds. Setting your polling interval to anything lower than 60 seconds will not speed up the errand.

The `polling_initial_offset` default is set to 5 seconds because we don't want the delete all errand to start polling cloudfoundry before cloudfoundry has finished processing the delete request and has contacted the broker. In systems with more load on cloudfoundry, this process could take a longer, in which case you might consider increasing the polling offset.

Run the errand with `bosh run errand delete-all-service-instances-and-deregister-broker`.

## Deregister Broker

This errand deregisters a broker from Cloud Foundry. It requires that there are no existing service instances.

Add the following instance group to your manifest:

```
- name: deregister-broker
lifecycle: errand
instances: 1
jobs:
  - name: deregister-broker
    release: <odb-release-name>
    properties:
      broker_name: <broker-name>
      disable_ssl_cert_verification: <truelfalse> # defaults to false
    cf:
      api_url: <cf-api-url>
      admin_username: <cf-api-admin-username>
      admin_password: <cf-api-admin-password>
vm_type: <vm-type>
stemcell: <stemcell>
networks: [{name: <service-network>}]
azs: [<az>]
```

Run the errand with `bosh run errand deregister-broker`.

## Delete All Service Instances

This errand deletes all service instances of your broker's service offering in every org and space of Cloud Foundry. It uses the Cloud Controller API to do this, and therefore only deletes instances the Cloud Controller knows about. It will not delete orphan BOSH deployments: those that don't correspond to a known service instance. Orphan BOSH deployments *should* never happen, but in practice they might. Use the [orphan-deployments errand](#) to identify them.

The errand will do the following operations:

1. Unbind all applications from the service instances
2. Delete all service instances sequentially
3. Check if any instances have been created while the errand was running

- a. If instances are detected the errand will fail
- b. Re-run the errand

This errand should only be used with extreme caution when you want to totally destroy all of the on-demand service instances from Cloud Foundry.

Add the following instance group to your manifest:

```
- name: delete-all-service-instances
  lifecycle: errand
  instances: 1
  jobs:
    - name: delete-all-service-instances
      release: <odb-release-name>
      properties:
        polling_interval: <interval in seconds when waiting for service instance to be deleted> # defaults to 60
        polling_initial_offset: <offset in seconds before starting to poll Cloud Foundry to check if the instance has been deleted> # defaults to 5

  vm_type: <vm-type>
  stemcell: <stemcell>
  networks: [{name: <network>}]
  azs: [<az>]
```



#### Notes:

The `polling_interval` default is set to 60 seconds because the Cloud Controller itself polls the on-demand broker every 60 seconds. Setting your polling interval to anything lower than 60 seconds will not speed up the errand.

The `polling_initial_offset` default is set to 5 seconds because we don't want the delete all errand to start polling cloudfoundry before cloudfoundry has finished processing the delete request and has contacted the broker. In systems with more load on cloudfoundry, this process could take a longer, in which case you might consider increasing the polling offset.

Run the errand with `bosh run errand delete-all-service-instances`.

## Delete Orphaned Deployments

The deployment for a service instance is defined as 'Orphaned' when the Bosh deployment is still running, but the service is no longer registered in Cloud Foundry.

The `orphan-deployments` errand will collate a list of service deployments that have no matching service instances in Cloud Foundry and return the list to the operator. It is then up to the operator to remove the orphaned bosh deployments.

Run the errand with `bosh run errand orphan-deployments`.

If orphan deployments are present, the errand will output a list of deployment names

```
[stdout]
[{"deployment_name":"service-instance_aoeu39fgn-8125-05h2-9023-9vbx7676f3"}]

[stderr]
None

Errand 'orphan-deployments' completed successfully (exit code 0)
```

**Warning:** Deleting the bosh deployment will destroy the vm, any data present will be lost.

To delete the orphan deployment run `bosh delete deployment service-instance_aoeu39fgn-8125-05h2-9023-9vbx7676f3`.

## Updates

### Update Broker

To update the [core broker configuration](#):

- make any necessary changes to the core broker configuration in the broker manifest
- deploy the broker

## Update Service Offering

To update the service offering:

- make any changes to properties in the `service_catalog` of the broker manifest. For example, update the service metadata.
- make any changes to properties in the `service_deployment` of the broker manifest. For example, update the jobs used from a service release.
- deploy the broker

**Warning:** Once the broker has been registered with Cloud Foundry do not change the `service_id` or the `plan_id` for any plan. When the ODB starts it checks that all existing service instances in Cloud Foundry have a `plan_id` that exists in the `service_catalog`.

After changing the `service_catalog`, you should run the [register-broker errand](#) to update the Cloud Foundry marketplace.

When the plans are updated in the `service_catalog`, then upgrades will need to be applied to existing service instances. See [upgrading all service instances](#).

## Disable Service Plans

Access to a service plan can be disabled by using the Cloud Foundry CLI:

```
$ cf disable-service-access <service-name-from-catalog> -p <plan-name>
```

Also, when a plan has the property `cf_service_access: disable` in the `service_catalog` then the [register-broker errand](#) errand will disable service access to that plan.

## Remove Service Plans

A service plans can be removed if there are no instances using the plan. To remove the a plan, remove it from the broker manifest and update the Cloud Foundry marketplace by running the [register-broker errand](#).

**Warning:** If any service instances remain on a plan that has been removed from the catalog then the ODB will fail to start.

## Upgrades

### Upgrade the Broker

The broker is upgraded in a similar manner to all BOSH releases:

- upload new version of `on-demand-service-broker-release` BOSH release to the BOSH Director
- make any necessary changes to the core broker configuration in the broker manifest
- deploy the broker

### Upgrade Service Offering

The service offering consists of:

- service catalog
- service adapter BOSH release
- service BOSH release(s)
- service stemcell

To upgrade a service offering:

- make any changes to the service catalog in the broker manifest
- upload any new service BOSH release(s) to the BOSH Director
- make any changes to service release(s) in the broker manifest
- upload any new service stemcell to the BOSH Director
- make any changes to the service stemcell in the `service_deployment` broker manifest
- deploy the broker

Any new service instances will be created using the latest service offering.

To upgrade all existing instances you can run the [upgrade-all-service-instances errand](#).

**Warning:** Until a service instance has been upgraded, `cf update-service` operations will be blocked and an error will be shown, see [updating service offering](#).

## Upgrade an Individual Service Instance

Cloud Foundry users cannot upgrade their service instances to the latest service offering.

Until a service instance has been upgraded, Cloud Foundry users cannot set parameters, or change plan until the service instance has been upgraded by an operator:

```
$ cf update-service my-redis -c '{"maxclients": 10000}'
Updating service instance my-redis as admin...
FAILED
Server error, status code: 502, error code: 10001, message: Service broker error: Service cannot be updated at this time, please try again later or contact your operator for more information.
```

Operators should run the [upgrade-all-service-instances errand](#) to upgrade all service instances to the latest service offering.

## Upgrade All Service Instances

To upgrade all existing service instances after the service offering has been updated or upgraded:

1. Add the following instance group to your broker manifest:

```
- name: upgrade-all-service-instances
  lifecycle: errand
  instances: 1
  jobs:
    - name: upgrade-all-service-instances
      release: <odb-release-name>
      properties:
        polling_interval: <interval in seconds when waiting for service instance to be upgraded> # defaults to 60
      vm_type: <vm-type>
      stemcell: <stemcell>
      networks: [{name: <network>}]
      azs: [<az>]
```

2. Deploy the broker manifest.
3. Run the errand with `bosh run errand upgrade-all-service-instances`.

Note, the `upgrade-all-service-instances` errand will trigger [service instance lifecycle errands](#) configured for the broker.

## Security

### BOSH API Endpoints

The ODB accesses the following [BOSH API](#) endpoints during the service instance lifecycle:

API endpoint	Examples of usage in the ODB
POST /deployments	create, or update a service instance
POST /deployments/<deployment_name>/errands/<errand_name>/runs	register, or de-register the on-demand broker with the Cloud Controller, run smoke tests
GET /deployments/<deployment_name>	passed as argument to the service adapter for <code>generate-manifest</code> and <code>create-binding</code>
GET /deployments/<deployment_name>/vms?format=full	passed as argument to the service adapter for <code>create-binding</code>
DELETE /deployments/<deployment_name>	delete a service instance
GET /tasks/<task_ID>/output?type=result	check a task was successful (i.e. the exit code was zero), get list of VMs
GET /tasks/<task_ID>	poll the BOSH director until a task finishes, e.g. create, update, or delete a deployment
GET /tasks?deployment=<deployment_name>	determine the last operation status and message for a service instance, e.g. 'create in progress' - used when creating, updating, deleting service instances

## BOSH UAA Permissions

The actions that the ODB needs to be able to perform are:

Modify:

- `bosh deploy`
- `bosh delete deployment`
- `bosh run errand`

Read only:

- `bosh deployments`
- `bosh vms`
- `bosh tasks`

The minimum UAA authority required by the BOSH Director to perform these actions is `bosh.teams.<team>.admin`. Note: a team admin cannot view or update the director's cloud config, nor upload releases or stemcells.

For more details on how to set up and use BOSH teams, see [Director teams and permissions configuration](#).

## Unused BOSH permissions

The team admin authority also allows the following actions, which currently are not used by the ODB:

- `bosh start/stop/recreate`
- `bosh cck`
- `bosh ssh`
- `bosh logs`
- `bosh releases`
- `bosh stemcells`

## PCF IPsec Add-On

The ODB has been tested with the [PCF IPsec Add-On](#), and it appears to work. Note that we excluded the BOSH director itself from IPsec ranges, as the BOSH add-on cannot be applied to BOSH itself.

## Troubleshooting

## Administer Service Instances

We recommend using the [bosh cli gem](#) for administering the deployments created by ODB; for example for checking VMs, ssh, viewing logs.

We **recommend against** using the bosh cli for updating/deleting ODB service deployments as it might accidentally trigger a race condition with Cloud Controller-induced updates/deletes or result in ODB overriding your [snowflake](#) changes at the next deploy. All updates to the service instances must be done using the [errand to upgrade all service instances](#).

## Logs

The on-demand broker writes logs to a log file, and to syslog.

The broker log contains error messages and non-zero exit codes returned by the service adapter, as well as the stdout and stderr streams of the adapter.

The log file is located at `/var/vcap/sys/log/broker/broker.log`. In syslog, logging is written with the tag `on-demand-service-broker`, under the facility `user`, with priority `info`.

If you want to forward syslog to a syslog aggregator, we recommend co-locating [syslog release](#) with the broker.

The ODB generates a UUID for each request and prefixes all the logs for that request, e.g.

```
on-demand-service-broker: [on-demand-service-broker] [4d63080d-e038-45a3-85f9-93910f6b40b1] 2016/09/05 16:43:26.123456 a valid UAA token was found in cache, will not obtain a new one
```

NB: The ODB's negroni server and start up logs are not prefixed with a request ID.

All ODB's logs have a UTC timestamp.

## Metrics

If you have [configured broker metrics](#), the broker will emit metrics to the CF firehose. You can, for example, consume these metrics by using the [CF CLI firehose plugin](#).

### Service-level Metrics

The broker will emit a metric indicating the total number of instances across all plans. In addition, if there is a global quota set for the service, a metric showing how much of that quota is remaining will be emitted. Service-level metrics use the format shown below.

```
origin:"<broker deployment name>" eventType:ValueMetric timestamp:<timestamp> deployment:"<broker deployment name>" job:"broker" index:"<bosh job index>" ip:"<IP>" valueMetric:<name>"/on-demand-
origin:"<broker deployment name>" eventType:ValueMetric timestamp:<timestamp> deployment:"<broker deployment name>" job:"broker" index:"<bosh job index>" ip:"<IP>" valueMetric:<name>"/on-demand-
```

### Plan-level Metrics

For each service plan, the metrics will report the total number of instances for that plan. If there is a quota set for the plan, the metrics will also report how much of that quota is remaining. Plan-level metrics are emitted in the following format.

```
origin:"<broker deployment name>" eventType:ValueMetric timestamp:<timestamp> deployment:"<broker deployment name>" job:"broker" index:"<bosh job index>" ip:"<IP>" valueMetric:<name>"/on-demand-
origin:"<broker deployment name>" eventType:ValueMetric timestamp:<timestamp> deployment:"<broker deployment name>" job:"broker" index:"<bosh job index>" ip:"<IP>" valueMetric:<name>"/on-demand-
```

If `quota_remaining` is `0` then you need to increase your plan quota in the BOSH manifest.

## Identify Deployments in BOSH

There is a one to one mapping between the service instance id from CF and the deployment name in BOSH. The convention is the BOSH deployment name would be the service instance id prepended by `service-instance_`. To identify the BOSH deployment for a service instance you can.

1. Determine the GUID of the service



```
$ cf service --guid <service-name>
```

2. Identify deployment in `bosh deployments` by looking for `service-instance-<GUID>`
3. Get current tasks for the deployment by using

```
$ bosh tasks --deployment service-instance-<GUID>
```

## Identify Tasks in BOSH

Most operations on the on demand service broker API are implemented by launching BOSH tasks. If an operation fails, it may be useful to investigate the corresponding BOSH task. To do this:

1. Determine the ID of the service for which an operation failed. You can do this using the Cloud Foundry CLI:

```
$ cf service --guid <service name>
```

2. SSH on to the service broker VM:

```
$ bosh deployment <path to broker manifest>
$ bosh ssh
```

3. In the broker log, look for lines relating to the service, identified by the service ID. Lines recording the starting and finishing of BOSH tasks will also have the BOSH task ID:

```
on-demand-service-broker: [on-demand-service-broker] [4d63080d-e038-45a3-85f9-93910f6b40b1] 2016/04/13 09:01:50.793965 Bosh task id for Create instance 30d4a67f-d220-4d06-9989-58a976b86b35
on-demand-service-broker: [on-demand-service-broker] [4d63080d-e038-45a3-85f9-93910f6b40b1] 2016/04/13 09:06:55.793976 task 11470 success creating deployment for instance 30d4a67f-d220-4d06-9989-58a976b86b35

on-demand-service-broker: [on-demand-service-broker] [8bf5c9f6-7acd-4ab4-9214-363a6f6bef79] 2016/04/13 09:16:20.795035 Bosh task id for Update instance 30d4a67f-d220-4d06-9989-58a976b86b35
on-demand-service-broker: [on-demand-service-broker] [8bf5c9f6-7acd-4ab4-9214-363a6f6bef79] 2016/04/13 09:17:20.795181 task 11473 success updating deployment for instance 30d4a67f-d220-4d06-9989-58a976b86b35

on-demand-service-broker: [on-demand-service-broker] [af6fab15-c95e-438b-aa6b-bc4329d4154f] 2016/04/13 09:17:52.803824 Bosh task id for Delete instance 30d4a67f-d220-4d06-9989-58a976b86b35
on-demand-service-broker: [on-demand-service-broker] [af6fab15-c95e-438b-aa6b-bc4329d4154f] 2016/04/13 09:19:56.803938 task 11474 success deleting deployment for instance 30d4a67f-d220-4d06-9989-58a976b86b35
```

4. Use the task ID to obtain the task log from BOSH (adding flags such as `--debug` or `--cpi` as necessary):

```
$ bosh task <task_ID>
```

## Identify Issues When Connecting to BOSH or UAA

The ODB interacts with the BOSH director to provision and deprovision instances, and is authenticated via the director's UAA. See [Core Broker Configuration](#) for an example configuration.

If BOSH and/or UAA are wrongly configured in the broker's manifest, then meaningful error messages will be displayed in the broker's log, indicating whether the issue is caused by an unreachable destination or bad credentials.

For example

```
on-demand-service-broker: [on-demand-service-broker] [575afbc1-b541-481d-9cde-b3d3e67e87bf] 2016/05/18 15:56:40.100579 Error authenticating (401): {"error":"unauthorized","error_description":"Bad credentials"}
```

## List Service Instances

The ODB persists the list of ODB-deployed service instances and provides an endpoint to retrieve them. This endpoint requires basic authentication.

During disaster recovery this endpoint could be used to assess the situation.

### Request

```
GET http://username:password@<ON_DEMAND_BROKER_IP>:8080/mgmt/service_instances
```

## Response

200 OK

Example JSON body:

```
[
  {
    "instance_id": "4d19462c-33cf-11e6-91cc-685b3585cc4e",
    "plan_id": "60476620-33cf-11e6-a841-685b3585cc4e",
    "bosh_deployment_name": "service-instance_4d19462c-33cf-11e6-91cc-685b3585cc4e"
  },
  {
    "instance_id": "57014734-33cf-11e6-ba8d-685b3585cc4e",
    "plan_id": "60476620-33cf-11e6-a841-685b3585cc4e",
    "bosh_deployment_name": "service-instance_57014734-33cf-11e6-ba8d-685b3585cc4e"
  }
]
```

## List Orphan Deployments

The On-Demand Broker provides an endpoint that compares the list of service instance deployments against the service instances registered in Cloud Foundry. When called, the endpoint will return a list of orphaned deployments, if any are present.

This endpoint is exercised in the `orphan-deployments` errand. To call this endpoint without running the errand, use curl

**Request** `GET http://username:password@<ON_DEMAND_BROKER_IP>:8080/mgmt/orphan_deployments`

## Response

200 OK

Example JSON body:

```
[
  {
    "deployment_name": "service-instance_d482abd3-8051-48d2-8067-9ccdf02327f3"
  }
]
```

## Backup and Restore Considerations

### On-Demand Service Broker

The on-demand service broker is stateless, so there is nothing to backup or restore.

### On-Demand Service Instances

Service instances created by the on-demand service broker may have state that needs to be backed up, e.g. data services.

It is the responsibility of the Service Author to provide documentation for the operator to backup and restore on-demand service instances. For a list of deliverables provided by the Service Author, see [Required Deliverables](#).

### Disaster Recovery

The on-demand service broker fetches the state of service instances and their deployments from the Cloud Foundry API and BOSH Director respectively. Therefore, to recover on-demand service instances in a disaster both the Cloud Controller database and BOSH Director database must be restored from a backup.

- [Backing Up and Restoring Pivotal Cloud Foundry](#) [↗](#)
- [How to backup and restore a BOSH Director deployment](#) [↗](#)

## Troubleshooting Errors

Start here if you're responding to a specific errors or error messages.

### Failed Install

1. Certificate issues: The on-demand broker (ODB) requires valid certificates. Ensure that your certificates are valid and [generate new ones](#) if necessary.
2. Deploy fails: Deploys can fail for a variety of reasons. View the logs using Ops Manager to determine why the deploy is failing.
3. [Networking problems](#):
  - Cloud Foundry cannot reach the on-demand service broker
  - Cloud Foundry cannot reach the service instances
  - The service network cannot access the BOSH director
4. [Register broker errand](#) fails.
5. The smoke test errand fails.
6. Resource sizing issues: These occur when the resource sizes selected for a given plan are less than the on-demand service requires to function. Check your resource configuration in Ops Manager and ensure that the configuration matches that recommended by the service.
7. Other service-specific issues.

### Cannot Create or Delete Service Instances

If developers report errors such as the following:

```
Instance provisioning failed: There was a problem completing your request. Please contact your operations team providing the following information: service: redis-acceptance, service-instance-g
```

[Log in to BOSH](#) and target the on-demand service instance using the instructions on [parsing a Cloud Foundry error message](#).

Retrieve the BOSH task ID from the error message and run `bosh task TASK-ID`.

If the BOSH error shows a problem with the deployment manifest:

1. Download the manifest for the on-demand service instance by running `bosh download manifest service-instance_SERVICE-INSTANCE-GUID MY-SERVICE.yml`.
2. Check the manifest for configuration errors.

[Access the broker logs](#) and use the `broker-request-id` from the error message above to search the log for more information.

Check for:

- [Authentication errors](#)
- [Network errors](#)
- [Quota errors](#)

### Broker Request Timeouts

If developers report errors such as:

```
Server error, status code: 504, error code: 10001, message: The request to the service broker timed out: https://BROKER-URL/v2/service_instances/e34046d3-2379-40d0-a318-d54fc7a5b13f/ser
```

1. Validate that Cloud Foundry (CF) has [network connectivity to the service broker](#).
2. Check the BOSH queue size:

- Log into BOSH as the admin user
- Run `bosh tasks`

3. If there are a large number of queued tasks then the system might be under load. BOSH is configured with two workers and one status worker. Advise app developers to try again later once the system is under less load.

In the future, Ops Manager will support configuring the number of BOSH workers available to the system.

## Cannot Bind to or Unbind from Service Instances

### Instance Does Not Exist

If developers report errors such as:

```
Server error, status code: 502, error code: 10001, message: Service broker error: instance does not exist`
```

1. Run `cf service MY-INSTANCE --guid` to check that the on-demand service instance exists in BOSH and CF.
2. Run `bosh vms service-instance_GUID` with the GUID you just obtained.

If the bosh deployment is not found then it has been deleted from BOSH. Contact Pivotal support for further assistance on recovery.

### Other Errors

If developers report errors such as:

```
Server error, status code: 502, error code: 10001, message: Service broker error: There was a problem completing your request. Please contact your operations team providing the following information
```

To find out the exact issue with the binding process:

1. [Access the service broker logs](#).
2. Search the logs for the `broker-request-id` string listed in the error message above.
3. Contact Pivotal support for further assistance if you are unable to resolve the problem.
4. Check for:
  - [Authentication errors](#)
  - [Network errors](#)

## Cannot Connect to a Service Instance

If developers report that their app cannot use service instances that they have successfully created and bound:

Ask the user to send application logs that show the connection error. If the error is originating from the service, then follow service-specific instructions. If the issue appears to be network-related, then:

1. Check that [application security groups](#) are configured correctly. Access should be configured for the service network that the tile is deployed to.
2. Ensure that the network the Elastic Runtime or Pivotal Application Service (PAS) tile is deployed to has network access to the service network. You can find the network definition for this service network in the Ops Manager Director tile.
3. From the Ops Manager Installation Dashboard, navigate to the service tile to see that the service network is configured in the **Networks** section.
4. From the Ops Manager Installation Dashboard, navigate to the Elastic Runtime (or PAS) tile and see the network it is assigned to. Make sure that these networks can access each other.

## Upgrade All Instances Fails

If the `upgrade-all-service-instances` errand fails, look at the errand output in the Ops Manager log.

If an instance fails to upgrade, debug and fix it before running the errand again to prevent any failure issues from spreading to other on-demand instances.

Once the Ops Manager log no longer lists the deployment as `failing`, [re-run the errand](#) to upgrade the rest of the instances.

## Missing Logs and Metrics

If no logs are being emitted by the on-demand broker, check that your syslog forwarding address is correct in Ops Manager.

1. Ensure you have configured syslog for the tile.
2. Ensure that you have network connectivity between the networks that the tile is using and the syslog destination. If the destination is external, you need to use the [public ip](#) VM extension feature available in your Ops Manager tile configuration settings.
3. Verify that the Firehose is emitting metrics:

- a. Install the `cf nozzle` [plugin](#).
- b. Run `cf nozzle -f ValueMetric | grep --line-buffered "on-demand-broker/MY-SERVICE"` to find logs from your service in the `cf nozzle` output.

If no metrics appear within five minutes, verify that the broker network has access to the Loggregator system on all required ports.

[Contact Pivotal support](#) if you are unable to resolve the issue.

## Knowledge Base (Community)

Find the answer to your question and browse product discussions and solutions by searching the [Pivotal Knowledge Base](#).

## File a Support Ticket

You can file a support ticket [here](#). Be sure to provide the error message from `cf service YOUR-SERVICE-INSTANCE`.

To help expedite troubleshooting, also provide your service broker logs, your service instance logs and BOSH task output, if your `cf service YOUR-SERVICE-INSTANCE` output includes a `task-id`.

## Troubleshooting Components

Guidance on checking for and fixing issues in the ODB components.

### BOSH problems

#### Missing BOSH Director UUID

If using the BOSH v1 CLI you need to re-add the `director_uuid` to the manifest:

1. Run `bosh status --uuid` and record the `director_uuid` value from the output.
2. Edit the manifest and add the `director_uuid: DIRECTOR-UUID` from the last step at the top of the manifest.

For more, see [Deployment Identification](#) in the BOSH docs.

#### Large BOSH Queue

On-demand service brokers add tasks to the BOSH request queue, which can back up and cause delay under heavy loads. An app developer who requests a new service instance sees `create in progress` in the Cloud Foundry Command Line Interface (cf CLI) until BOSH processes the queued request.

Ops Manager currently deploys two BOSH workers to process its queue. Future versions of Ops Manager will let users configure the number of BOSH workers.

## Configuration

#### Service instances in failing state


You might have configured a VM / Disk type in tile plan page in Ops Manager that is insufficiently large for the on-demand service instance to start. See tile-specific guidance on resource requirements.

## Authentication

#### UAA Changes

If you have rotated any UAA user credentials then you might see authentication issues in the service broker logs.

To resolve this, redeploy the service tile in Ops Manager. This provides the broker with the latest configuration.

 **Note:** You must ensure that any changes to UAA credentials are reflected in the OpsManager `credentials` tab of the Elastic Runtime or Pivotal Application Service (PAS) tile.

## Networking

Common issues include:

1. Network latency when connecting to the on-demand service instance to create or delete a binding.
  - Solution: Try again or improve network performance

2. Network firewall rules are blocking connections from the on-demand service broker to the service instance.
  - Solution: Open the service tile in Ops Manager and check the two networks configured in the **Networks** pane. Ensure that these networks allow access to each other.
3. Network firewall rules are blocking connections from the service network to the BOSH director network.
  - Solution: Ensure that service instances can access the Director so that the BOSH agents can report in.
4. Apps cannot access the service network.
  - Solution: Configure Cloud Foundry application security groups to allow runtime access to the service network.
5. Problems accessing BOSH's UAA or the BOSH director.
  - Solution: Follow network troubleshooting and check that the BOSH director is online.

## Validate Service Broker Connectivity to Service Instances

To validate you can `bosh ssh` onto the on-demand service broker, download the broker manifest and target the deployment, then try to reach the service instance.

If no BOSH `task-id` appears in the error message, look in the broker log using the `broker-request-id` from the task.

## Validate App Access to Service Instance

Use `cf ssh` to access to the app container, then try connecting to the on-demand service instance using the binding included in the `VCAP_SERVICES` environment variable.

## Quotas

### Plan Quota issues

If developers report errors such as:

Message: Service broker error: The quota for this service plan has been exceeded.  
Please contact your Operator for help.

1. Check your current plan quota.
2. Increase the plan quota.
3. Log into Ops Manager.
4. Reconfigure the quota on the plan page.
5. Deploy the tile.
6. Find who is using the plan quota and take the appropriate action.

### Global Quota Issues

If developers report errors such as:

Message: Service broker error: The quota for this service has been exceeded.  
Please contact your Operator for help.

1. Check your current global quota.



2. Increase the global quota.
3. Log into Ops Manager.
4. Reconfigure the quota on the on-demand settings page.
5. Deploy the tile.
6. Find out who is using the quota and take the appropriate action.

## Failing Jobs and Unhealthy Instances

To determine whether there is an issue with the on-demand service deployment, run `bosh vms`:

```
$ bosh vms --vitals service-instance_$guid
```

For additional information, run `bosh instances`:

```
$ bosh instances --ps --vitals
```

If the VM is failing, follow the service-specific information. Any unadvised corrective actions (such as running `bosh restart` on a VM) might cause issues in the service instance.

## Knowledge Base (Community)

Find the answer to your question and browse product discussions and solutions by searching the [Pivotal Knowledge Base](#).

## File a Support Ticket

You can file a support ticket [here](#). Be sure to provide the error message from `cf service YOUR-SERVICE-INSTANCE`.

To help expedite troubleshooting, also provide your service broker logs, your service instance logs and BOSH task output, if your

`cf service YOUR-SERVICE-INSTANCE` output includes a `task-id`.

## Techniques for Troubleshooting

Instructions on interacting with the on-demand service broker and on-demand service instance BOSH deployments, and on performing general maintenance and housekeeping tasks

### Parse a Cloud Foundry (CF) Error Message

Failed operations (create, update, bind, unbind, delete) result in an error message. You can retrieve the error message later by running the cf CLI command `cf service INSTANCE-NAME`.

```
$ cf service myservice

Service instance: myservice
Service: super-db
Bound apps:
Tags:
Plan: dedicated-vm
Description: Dedicated Instance
Documentation url:
Dashboard:

Last Operation
Status: create failed
Message: Instance provisioning failed: There was a problem completing your request.
Please contact your operations team providing the following information:
  service: redis-acceptance,
  service-instance-guid: ae9e232c-0bd5-4684-af27-1b08b0c70089,
  broker-request-id: 63da3a35-24aa-4183-aec6-db8294506bac,
  task-id: 442,
  operation: create
Started: 2017-03-13T10:16:55Z
Updated: 2017-03-13T10:17:58Z
```

Use the information in the `Message` field to debug further. Provide this information to Pivotal Support when filing a ticket.

The `task-id` field maps to the BOSH task id. For further information on a failed BOSH task, use the `bosh task TASK-ID` command in the BOSH CLI.

The `broker-request-guid` maps to the portion of the On-Demand Broker log containing the failed step. Access the broker log through your syslog aggregator, or access BOSH logs for the broker by typing `bosh logs broker 0`. If you have more than one broker instance, repeat this process for each instance.

## Access Broker and Instance Logs and VMs

Before following the procedures below, log into the [cf CLI](#) and the [BOSH CLI](#).

### Access Broker Logs and VM(s)

1. Run `bosh deployments` to identify the on-demand broker (ODB) deployment.
2. Run `bosh download manifest ODB-DEPLOYMENT-NAME odb.yml` to download the ODB manifest.
3. Select the ODB deployment using `bosh deployment odb.yml`.
4. View VMs in the deployment using `bosh instances`.
5. Run `bosh ssh INSTANCE-ID` to SSH onto the VM.
6. Run `bosh logs INSTANCE-ID` to download broker logs.

You can also [access logs using Ops Manager](#) by clicking on the **Logs** tab in the tile and downloading the broker logs.


The archive generated by BOSH or Ops Manager includes the following logs:

Log Name	Description
	Requests to the on-demand broker and the actions the broker performs while orchestrating the request (e.g. generating a manifest

Log Name	Description
broker.log	and calling BOSH). Start here when troubleshooting.
broker_ctl.log	Control script logs for starting and stopping the on-demand broker.
post-start.stderr.log	Errors that occur during post-start verification.
post-start.stdout.log	Post-start verification.

## Access Service Instance Logs and VMs

1. To target an individual service instance deployment, retrieve the GUID of your service instance with the cf CLI command `cf service MY-SERVICE --guid`.
2. Run `bosh status --uuid` to retrieve the BOSH Director GUID.

 **Note:** “GUID” and “UUID” mean the same thing.

3. To download your BOSH manifest for the service, run `bosh download manifest service-instance_SERVICE-INSTANCE-GUID MY-SERVICE.yml` using the GUID you just obtained and a filename you want to save the manifest as.
4. Edit the following line in the service instance manifest that you just saved, to include the current BOSH Director GUID:

```
director_uuid: BOSH-DIRECTOR-GUID
```

5. Run `bosh deployment MY-SERVICE.yml` to select the deployment using the Director UUID.
6. Run `bosh instances` to view VMs in the deployment.
7. Run `bosh ssh INSTANCE-ID` to SSH onto the VM.
8. Run `bosh logs INSTANCE-ID` to download instance logs.

## Run Service Broker Errands to Manage Brokers and Instances

From the BOSH CLI, you can run service broker errands that manage the service brokers and perform mass operations on the service instances that the brokers created. These service broker errands include:

- `register-broker` registers a broker with the Cloud Controller and lists it in the Marketplace
- `deregister-broker` deregisters a broker with the Cloud Controller and removes it from the Marketplace
- `upgrade-all-service-instances` upgrades existing instances of a service to its latest installed version
- `delete-all-service-instances` deletes all instances of service
- `orphan-deployments` detects “orphan” instances that are running on BOSH but not registered with the Cloud Controller

Before running any of these errands, set the service broker manifest in the BOSH CLI by doing the following:

1. Run `bosh deployments`.
2. In the **Name** column of the output, look for `grep` for a string of the form `p-rabbit-GUID`. This is the unique identifier for the broker in BOSH.
3. Run `bosh download manifest RABBIT-BROKER-GUID BROKER-MANIFEST.yml` with the unique string from the previous step and any filename you want to give the broker deployment manifest.
4. Run `bosh deployment BROKER-MANIFEST.yml` to select the broker deployment as the one to run broker errands against.

## Register Broker

This errand registers the broker with Cloud Foundry and enables access to plans in the service catalog. The errand should be run whenever the broker is re-deployed with new catalog metadata to update the Cloud Foundry catalog.

Plans with disabled service access are not visible to non-admin Cloud Foundry users (including Org Managers and Space Managers). Admin Cloud Foundry users can see all plans including those with disabled service access.

The errand does the following:

- Registers the service broker with Cloud Controller
- Enables service access for any plans that have the radio button set to `enabled` in the tile plan page.
- Disables service access for any plans that have the radio button set to `disabled` in the tile plan page.
- Does nothing for any for any plans that have the radio button set to `manual`

Run this errand with the command `bosh run errand register-broker` after you have selected the broker deployment with `bosh deployment`.

## Deregister Broker

This errand deregisters a broker from Cloud Foundry. It requires that there are no existing service instances. You can use the [Delete All Service Instances errand](#) to delete any existing service instances that are problematic.

The errand does the following:

- Deletes the service broker from Cloud Controller
- Fails if there are any service instances, with or without bindings

Run this errand with the command `bosh run errand deregister-broker` after you have selected the broker deployment with `bosh deployment`.

## Upgrade All Service Instances

If you have made changes to the plan definition or uploaded a new tile into OpsManager you might want to upgrade all the on-demand service instances to the latest software / plan definition.

To do this you can either select the errand through the OpsManager UI and have this happen along with the `apply-changes` action or run the errand directly with the command `bosh run errand upgrade-all-service-instances` after you have selected the broker deployment with `bosh deployment`.

The errand does the following:

- Collects all of the service instances the on-demand broker has registered.
- For each instance the errand serially:
  - Issues an upgrade command to the on-demand broker.
  - Re-generates the service instance manifest based on its latest configuration from the tile.
  - Deploys the new manifest for the service instance.
  - Waits for this operation to complete, then proceeds to the next instance.
- Adds to a retry list any instances that have ongoing BOSH tasks at the time of upgrade.
- Retries any instances in the retry list until all are upgraded.


If any instance fails to upgrade, the errand fails immediately. This prevents systemic problems from spreading to the rest of your service instances.

## Delete All Service Instances

This errand deletes all service instances of your broker's service offering in every org and space of Cloud Foundry. It uses the Cloud Controller API to do this, and therefore only deletes instances the Cloud Controller knows about. It will not delete orphan BOSH deployments: those that don't correspond to a known service instance. Orphan BOSH deployments should never happen, but in practice they might. Use the `orphan-deployments` errand to identify them.

The errand does the following:

- Unbinds all applications from the service instances.
- Deletes all service instances sequentially.
- Checks if any instances have been created while the errand was running.
- If newly-created instances are detected, the errand fails.

 **WARNING:** This errand should only be used with extreme caution when you want to totally destroy all of the on-demand service instances in an environment.

Run this errand with the command `bosh run errand delete-all-service-instances` after you have selected the broker deployment with `bosh deployment`.

## Detect Orphaned Instances Service Instances

A service instance is defined as 'orphaned' when the BOSH deployment for the instance is still running, but the service is no longer registered in Cloud Foundry.

The `orphan-deployments` errand collates a list of service deployments that have no matching service instances in Cloud Foundry and return the list to the operator. It is then up to the operator to remove the orphaned bosh deployments.

Run this errand with the command `bosh run errand orphan-deployments` after you have selected the broker deployment with `bosh deployment`.

If orphan deployments exist, the errand script will:

- Exit with exit code 10
- Output a list of deployment names under a `[stdout]` header
- Provide a detailed error message under a `[stderr]` header

For example:

```
[stdout]
[{"deployment_name":"service-instance_80e3c5a7-80be-49f0-8512-44840f3c4d1b"}]

[stderr]
Orphan BOSH deployments detected with no corresponding service instance in Cloud Foundry. Before deleting any deployment it is recommended to verify the service instance no longer exists in Cloud Foundry.

Errand 'orphan-deployments' completed with error (exit code 10)
```

These details will also be available through the BOSH `/tasks/` API endpoint for use in scripting:

```
$ curl 'https://bosh-user:bosh-password@bosh-url:25555/tasks/task-id/output?type=result' | jq .
{
  "exit_code": 10,
  "stdout": "[{"deployment_name":"service-instance_80e3c5a7-80be-49f0-8512-44840f3c4d1b"}]\n",
  "stderr": "Orphan BOSH deployments detected with no corresponding service instance in Cloud Foundry. Before deleting any deployment it is recommended to verify the service instance no longer exists in Cloud Foundry.",
  "logs": {
    "blobstore_id": "d830c4bf-8086-4be2-8c1d-54d3a3c6d88d"
  }
}
```

If no orphan deployments exist, the errand script will:

- Exit with exit code 0
- Stdout will be an empty list of deployments
- Stderr will be `None`

```
[stdout]
[]

[stderr]
None

Errand 'orphan-deployments' completed successfully (exit code 0)
```

If the errand encounters an error during running it will:

- Exit with exit 1
- Stdout will be empty
- Any error messages will be under stderr

To clean up orphaned instances, perform the following action on each:

```
$ bosh delete deployment service-instance_SERVICE-INSTANCE-GUID
```

**WARNING:** This might leave IaaS resources in an unusable state.

## Select the BOSH Deployment for a Service Instance

1. Retrieve the GUID of your service instance with command `cf service YOUR-SERVICE-INSTANCE --guid`.
2. To download your BOSH manifest for the service, run `bosh download manifest service-instance_SERVICE-INSTANCE-GUID myservice.yml` GUID you just obtained and a filename you want to save the manifest as.
3. Run `bosh deployment MY-SERVICE.yml` to select the deployment.

## Get Admin Credentials for a Service Instance

1. [Identify the service deployment by GUID.](#)
2. [Log into BOSH](#).
3. [Download the manifest for the service instance](#) and add the GUID if using the v1 BOSH CLI.
4. Look in the manifest for the credentials, as described in the service documentation.

## Reinstall a Tile

To reinstall a tile in the same environment where it was previously uninstalled:

1. Ensure that the previous tile was correctly uninstalled as follows:
  - a. `cf login` as an admin user.
  - b. Run `cf m` and confirm that the Marketplace does not list the service.
  - c. `bosh login` as an admin user.
  - d. Run `bosh deployments` and confirm that its output does not show a deployment for the service. For example no `p-concourse-GUID` deployment exists.
2. Run the `delete-all-service-instances` errand to delete all instances of the service.
3. Run the `deregister-broker` errand to delete the service broker.
4. Run `bosh delete deployment YOUR-BROKER-DEPLOYMENT` to delete the service broker BOSH deployment.
5. Install the tile again.

## View Resource Saturation and Scaling

To view usage statistics for any service, select the service broker deployment. Then run `bosh vms --vitals` and `bosh instances --vitals` to view current resource utilization.

You can also view process-level information by using `bosh instances --ps`.

## Identify Service Instance Owner


If you have spotted a failing service instance deployment, you can identify which org/space owns the instance and list the apps bound to it by following these steps:

1. `bosh vms service-instance_SERVICE-INSTANCE-GUID` shows a VM in a failing state.
2. Take the deployment name and strip the `service-instance_` leaving you with the GUID.
3. Login to CF as an admin.
4. Run `cf curl /v2/service_instances/GUID | grep "space_url"` and record the SPACE-GUID field from the output `"space_url": "/v2/spaces/SPACE-GUID"`.
5. run `cf curl /v2/spaces/SPACE-GUID | grep -E "name|organization_url"` with the SPACE-GUID above and record from the output:
  - a. The SPACE-NAME field from the output `"name": "MY-SPACE",` and
  - b. The ORG-GUID field from the output `"organization_url": "/v2/organizations/ORG-GUID"`.
6. Run `cf curl /v2/organizations/ORG-GUID | grep "name"` with the ORG-GUID above, and record the MY-ORG field from the output `"name": "MY-ORG"`.
7. Run `cf target -o MY-ORG -s MY-SPACE` with the values above to target the org and space.
8. Run `cf services` to see all the services and bound apps in the space.
9. Run `cf curl /v2/spaces/SPACE-GUID/managers` to find out who is the space manager.
10. Use this information to contact the space manager if needed.

## Monitor Quota Saturation and Service Instance Count

Quota saturation and total number of service instances are available through ODB metrics emitted to Loggregator. The metric names are shown below:

Metric Name	Description
<code>on-demand-broker/{service-name-marketplace}/quota_remaining</code>	global quota remaining for all instances across all plans
<code>on-demand-broker/{service-name-marketplace}/{plan_name}/quota_remaining</code>	quota remaining for a particular plan
<code>on-demand-broker/{service-name-marketplace}/total_instances</code>	total instances created across all plans
<code>on-demand-broker/{service-name-marketplace}/{plan_name}/total_instances</code>	total instances created for a given plan

 **Note:** Quota metrics are not emitted if no quota has been set.

## Knowledge Base (Community)

Find the answer to your question and browse product discussions and solutions by searching the [Pivotal Knowledge Base](#).

## File a Support Ticket

You can file a support ticket [here](#). Be sure to provide the error message from `cf service YOUR-SERVICE-INSTANCE`.

To help expedite troubleshooting, also provide your service broker logs, your service instance logs and BOSH task output, if your `cf service YOUR-SERVICE-INSTANCE` output includes a `task-id`.

## Creating an On-Demand Service Tile

This documents the process for deploying an on-demand broker (ODB) with a service in a single tile, on a AWS installation of Ops Manager 1.8. We have built a reference [Kafka tile](#).

## Requirements

Before ODB, Ops Manager controlled the IP allocation of the private networks. So when using the ODB in a tile, you will need at least two private networks:

- a network where Ops Manager will deploy the on-demand broker VM
- a different network where the on-demand broker will deploy service instance VMs

The network for service instances should be flagged as a Service Network in Ops Manager.

## Deploying Ops Manager to AWS

1. Follow the default Ops Manager deployment [docs](#), but with these modifications:
  - a. Create a self-signed wildcard SSL certificate for a domain you control: This will usually be `*.some-subdomain.cf-app.com`, and upload it (along with the associated private key) to AWS. Instructions [here](#).
  - b. Download the CloudFormation JSON and save it in the Ops Manager directory.
  - c. Run the CloudFormation stack, saving any pertinent inputs (e.g BOSH DB credentials) you type into the web console into the Ops Manager directory for safe keeping (e.g. in `info.txt`).
  - d. Launch an instance of the AMI. If possible, use an elastic IP so that we can always keep the same DNS record even if we recreate the VM. Failing that, auto-assign a public IP.
  - e. Create a DNS record for `pcf.<the domain you made a wildcard cert for earlier>`. To use the earlier example, the record will be for `pcf.some-subdomain.cf-app.com`. It should point to the public IP of the Ops Manager VM.
2. Keep following the docs to log into Ops Manager (save the credentials).
3. Configure the Ops Manager Director (BOSH) tile.
4. Click “Apply Changes”, and steal the BOSH init manifest for future reference.

```
scp -i private_key.pem ubuntu@opsmanIP:/var/tempest/workspaces/default/deployments/bosh.yml bosh.yml
```

Notes:

1. The ELBs created by CloudFormation are both for CF, not Ops Manager. One of them will be configured with your wildcard certificate. This takes the place of HAProxy in AWS PCF deployments, and is therefore not used until you deploy the ERT tile.
2. To target the BOSH Director from the Ops Manager VM:

```
bosh --ca-cert /var/tempest/workspaces/default/root_ca_certificate target 10.0.16.10
```

## Build a Tile

Follow the default build your own [Product tile documentation](#), enhance the `handcraft.yml` with the accessors listed below. To access the `$self` accessors, the `service-broker` flag must be `true` in the handcraft.

## Non-Exhaustive Accessors Reference

director

Used to provide fields relating to the BOSH Director installation present.

Accessor	Description
<code>\$director.hostname</code>	The director’s hostname or IP address



Accessor	Description
<code>director.ca_public_key</code>	Director's root ca certificate. Related: <a href="#">how to configure SSL certificates for the ODB</a> .

For example

```
bosh:
url: https://(( $director.hostname )):25555
root_ca_cert: (( $director.ca_public_key ))
```

## self

Used to provide fields that belong to the specific tile (in this case, the broker tile).

Accessor	Description
<code>\$self.uaa_client_name</code>	UAA client name, that can authenticate with the BOSH Director
<code>\$self.uaa_client_secret</code>	UAA client secret, that can authenticate with the BOSH Director
<code>\$self.service_network</code>	Service network configured for the on-demand instances

The service network has to be created manually. Create a subnet on AWS and then add it to the director. In the director tile, under Create Networks > ADD network > fill in the subnet/vpc details.

`$self` accessors are enabled by setting `service_broker: true` at the top level of `handcraft.yml`. Please note that, at the time of writing this, setting `service_broker: true` will cause a redeployment of the BOSH Director when installing or uninstalling the tile.

For example

```
bosh:
authentication:
  uaa:
    url: https://(( $director.hostname )):8443
    client_id: (( $self.uaa_client_name ))
    client_secret: (( $self.uaa_client_secret ))
```

## cf

Used to provide fields from the Elastic Runtime Tile (i.e. Cloud Foundry) present in the Ops Manager installation.

Accessor	Description
<code>..cf.ha_proxy.skip_cert_verify.value</code>	Flag to skip SSL certificate verification for connections to the CF API
<code>..cf.cloud_controller.apps_domain.value</code>	The application domain configured in the CF installation
<code>..cf.cloud_controller.system_domain.value</code>	The system domain configured in the CF installation
<code>..cf.uaa.system_services_credentials.identity</code>	Username of a CF user in the cloud_controller.admin group, to be used by services
<code>..cf.uaa.system_services_credentials.password</code>	Password of a CF user in the cloud_controller.admin group, to be used by services

For example

```
disable_ssl_cert_verification: (( ..cf.ha_proxy.skip_cert_verify.value ))
cf:
url: https://api.(( ..cf.cloud_controller.system_domain.value ))
authentication:
url: https://uaa.(( ..cf.cloud_controller.system_domain.value ))
user_credentials:
  username: (( ..cf.uaa.system_services_credentials.identity ))
  password: (( ..cf.uaa.system_services_credentials.password ))
```

## Reference

For more accessors you can see the [ops-manager-example product](#) [↗](#)

## Public IP address for on-demand service instance groups

Ops Manager 1.9 RC1+ provides a VM extension called `public_ip` in the BOSH Director's cloud config. This can be used in the on-demand service broker's manifest to give instance groups a public IP address. This IP is only used for outgoing traffic to the internet from VMs with the `public_ip` extension. All internal traffic / incoming connections need to go over the private IP.

Here is an example showing how to allow operators to assign a public IP address to an on-demand service instance group in the tile handcraft:

```
form_types:
- name: example_form
  property_inputs:
  - reference: .broker.example_vm_extensions
    label: VM options
    description: List of VM options for Service Instances

job_types:
- name: broker
  templates:
  - name: broker
    release: on-demand-service-broker
    manifest: |
      service_catalog:
        plans:
        - name: example-plan
          instance_groups:
          - name: example-instance-group
            vm_extensions: (( .broker.example_vm_extensions.value ))
  property_blueprints:
  - name: example_vm_extensions
    type: multi_select_options
    configurable: true
    optional: true
    options:
    - name: "public_ip"
      label: "Internet Connected VMs (on supported IaaS providers)"
```

## Floating stemcells

Ops Manager provides a feature called [Floating Stemcells](#) that allows PCF to quickly propagate a patched stemcell to all VMs in the deployment that have the same compatible stemcell. Both the broker deployment and the service instances deployed by the On-Demand Broker can make use of this feature. Enabling this feature can help ensure that all of your service instances are patched to the latest stemcell.

In order for the service instances to be installed automatically with the latest stemcell, you will need to make sure the `upgrade-all-service-instances` errand is ticked.

Here is an example of how to implement floating stemcells in `handcraft.yml`:

```
job_types:
  templates:
  - name: broker
    manifest: |
      service_deployment:
        releases:
        - name: release-name
          version: 1.0.0
        jobs: [job_server]
      stemcell:
        os: ubuntu-trusty
        version: (( $self.stemcell_version ))
```

Here is an example of how to configure the `stemcell_criteria` in `binaries.yml`:

```
---
name: example-on-demand-service
product_version: 1.0.0
stemcell_criteria:
  os: ubuntu-trusty
  version: '3312'
  enable_patch_security_updates: true
```

Please note, configuring this value to `false` will disable this feature.

## On-Demand Broker errands

In the reference [Kafka tile](#) [↗](#) we have demonstrated how the errands in the on-demand broker release can be used.

The errands should be specified in the following order, as shown in the example Kafka tile:

Post-deploy:

- `register-broker`
- `upgrade-all-service-instances`

Pre-delete:

- `delete-all-service-instances-and-deregister-broker`

These errands are documented in the [operating section](#).

## How On-Demand Services Process Commands

These sequence diagrams in this topic show how an on-demand service sets up and maintains service instances, indicating which tasks are undertaken by the on-demand broker (ODB) and which require interaction with the Service Adapter.

### Register Service Broker with Cloud Foundry

```
sequenceDiagram
    User->>Cloud Controller:cf create-service-broker
    Cloud Controller->>On Demand Broker:GET catalog
    On Demand Broker->>Cloud Controller:catalog
    Cloud Controller->>User:OK
```

### Create Service Instance

Note that there are two ways this can fail: synchronously and asynchronously. When it fails synchronously, the Cloud Controller will subsequently delete the service according to its [orphan mitigation strategy](#). In the case when it fails asynchronously (e.g. while BOSH deploys the service instance), the Cloud Controller won't issue a delete request.

```
sequenceDiagram
    User->>Cloud Controller:cf create-service
    Cloud Controller->>On Demand Broker:POST instance (provision)
    On Demand Broker->>Service Adapter:generate-manifest
    Service Adapter->>On Demand Broker:manifest
    On Demand Broker->>BOSH:deploy
    BOSH->>On Demand Broker:accepted
    On Demand Broker->>Cloud Controller:accepted
    Cloud Controller->>User:create in progress loop until bosh task is complete
    Cloud Controller->>On Demand Broker:GET last operation
    On Demand Broker->>BOSH:GET deploy task
    BOSH->>On Demand Broker:task in progress
    On Demand Broker->>Cloud Controller:create in progress end
    Cloud Controller->>On Demand Broker:GET last operation
    On Demand Broker->>BOSH:GET task
    BOSH->>On Demand Broker:task done
    On Demand Broker->>Cloud Controller:create succeeded
    User->>Cloud Controller:cf service
    Cloud Controller->>User:create succeeded
```

### Delete Service Instance

In the delete service workflow the service adapter is not invoked.

```
sequenceDiagram
    User->>Cloud Controller:cf delete-service
    Cloud Controller->>On Demand Broker:DELETE instance
    On Demand Broker->>BOSH:delete deployment
    BOSH->>On Demand Broker:accepted
    On Demand Broker->>Cloud Controller:accepted
    Cloud Controller->>User:delete in progress loop until bosh task is complete
    Cloud Controller->>On Demand Broker:GET last operation
    On Demand Broker->>BOSH:GET task
    BOSH->>On Demand Broker:task in progress
    On Demand Broker->>Cloud Controller:delete in progress end
    Cloud Controller->>On Demand Broker:GET last operation
    On Demand Broker->>BOSH:GET task
    BOSH->>On Demand Broker:task done
    On Demand Broker->>Cloud Controller:delete succeeded
    User->>Cloud Controller:cf service
    Cloud Controller->>User:not found
```

### Create/Update Service Instance with Post-Deploy Errand

ODB will not report create/update succeeded to Cloud Foundry until both the deployment and post-deploy errand have completed successfully.

```
sequenceDiagram
    User->>Cloud Controller:cf create-service
    Cloud Controller->>On Demand Broker:POST instance (create)
    On Demand Broker->>Service Adapter:generate-manifest
    Service Adapter->>On Demand Broker:manifest
    On Demand Broker->>BOSH:deploy
    BOSH->>On Demand Broker:accepted
    On Demand Broker->>Cloud Controller:accepted
    Cloud Controller->>User:create in progress loop until bosh task is complete
    Cloud Controller->>On Demand Broker:GET last operation
    On Demand Broker->>BOSH:GET deploy task
    BOSH->>On Demand Broker:task in progress
    On Demand Broker->>Cloud Controller:create in progress end
    Cloud Controller->>On Demand Broker:GET last operation
    On Demand Broker->>BOSH:GET task
    BOSH->>On Demand Broker:task done
    On Demand Broker->>BOSH:run post-deploy errand
    BOSH->>On Demand Broker:accepted loop until bosh task errand is complete
    Cloud Controller->>On Demand Broker:GET last operation
    On Demand Broker->>BOSH:GET errand task
    BOSH->>On Demand Broker:task in progress
    On Demand Broker->>Cloud Controller:create in progress end
    Cloud Controller->>On Demand Broker:GET last operation
    On Demand Broker->>BOSH:GET errand task
    BOSH->>On Demand Broker:task done
    On Demand Broker->>Cloud Controller:create succeeded
    User->>Cloud Controller:cf service
    Cloud Controller->>User:create succeeded
```

### Delete Service Instance with Pre-Delete Errand

ODB will not report delete succeeded to Cloud Foundry until both the pre-delete errand and delete deployment have completed successfully.

```
sequenceDiagram
    User->>Cloud Controller:cf delete-service
    Cloud Controller->>On Demand Broker:DELETE instance
    On Demand Broker->>BOSH:run pre-delete errand
    BOSH->>On Demand Broker:accepted
    On Demand Broker->>Cloud Controller:accepted
    Cloud Controller->>User:delete in progress loop
```

until errand bosh task is complete Cloud Controller->>On Demand Broker:GET last operation On Demand Broker->>BOSH:GET errand task BOSH->>On Demand Broker:task in progress On Demand Broker->>Cloud Controller:delete in progress end Cloud Controller->>On Demand Broker:GET last operation On Demand Broker->>BOSH:GET errand task BOSH->>On Demand Broker:task done On Demand Broker->>BOSH:delete deployment BOSH->>On Demand Broker:accepted loop until delete deployment bosh task is complete Cloud Controller->>On Demand Broker: GET last operation On Demand Broker->>BOSH: GET delete deployment task BOSH->>On Demand Broker: task in progress On Demand Broker->>Cloud Controller: delete in progress end Cloud Controller->>On Demand Broker: GET last operation On Demand Broker->>BOSH: GET delete deployment task BOSH->>On Demand Broker: task done On Demand Broker->>Cloud Controller: delete succeeded User->>Cloud Controller:cf service Cloud Controller->>User: not found

## Update Service Instance

Updates can only proceed if the existing service instance is up-to-date. ODB calls `generate-manifest` on service adapter to determine whether there are any pending changes for the instance.

### When There Are Pending Changes

sequenceDiagram User->> Cloud Controller: cf update-service -c '{"some": "config"}' Cloud Controller->> On Demand Broker: PATCH instance (update) On Demand Broker->>BOSH:GET manifest BOSH->>On Demand Broker:previous manifest On Demand Broker->>Service Adapter: generate-manifest (without request parameters) Service Adapter->>On Demand Broker: manifest On Demand Broker->>Cloud Controller: update failed, pending changes detected Cloud Controller->>User: update failed, pending changes detected

### When There Are No Pending Changes

The manifest from the second call to `generate-manifest` is deployed.

sequenceDiagram User->> Cloud Controller: cf update-service -c '{"some": "config"}' Cloud Controller->> On Demand Broker: PATCH instance (update) On Demand Broker->>BOSH:GET manifest BOSH->>On Demand Broker:previous manifest On Demand Broker->>Service Adapter: generate-manifest (without request parameters) Service Adapter->>On Demand Broker: manifest On Demand Broker->>Service Adapter: generate-manifest (with request parameters) Service Adapter->>On Demand Broker: manifest On Demand Broker->>BOSH: deploy BOSH->>On Demand Broker: accepted On Demand Broker->>Cloud Controller: accepted Cloud Controller->>User: update in progress loop until bosh task is complete Cloud Controller->>On Demand Broker:GET last operation On Demand Broker->>BOSH:GET task BOSH->>On Demand Broker:task in progress On Demand Broker->>Cloud Controller:update in progress end Cloud Controller->>On Demand Broker:GET last operation On Demand Broker->>BOSH:GET task BOSH->>On Demand Broker:task done On Demand Broker->>Cloud Controller:update succeeded User->>Cloud Controller: cf service Cloud Controller->>User: update succeeded

## Bind

sequenceDiagram User->>Cloud Controller:cf bind-service Cloud Controller->>On Demand Broker:PUT binding On Demand Broker->>Service Adapter:create-binding Service Adapter->>On Demand Broker:binding credentials On Demand Broker->>Cloud Controller:binding credentials Cloud Controller->>User:OK

## Unbind

sequenceDiagram User->>Cloud Controller:cf unbind-service Cloud Controller->>On Demand Broker:DELETE binding On Demand Broker->>Service Adapter:delete-binding Service Adapter->>On Demand Broker:exit 0 On Demand Broker->>Cloud Controller:OK Cloud Controller->>User:OK

## Upgrade All Service Instances

ODB provides BOSH errand to upgrade all the instances managed by the broker. This can also be used in the scenario when a plan changes; this errand will update all instances that implement the plan with the new plan definition.

sequenceDiagram Operator->>Upgrade Errand:bosh run errand upgrade-all-service-instances Upgrade Errand->>On Demand Broker:GET instances On Demand Broker->>Cloud Controller:GET instances Cloud Controller->>Upgrade Errand:instances loop for all instances Upgrade Errand->>On Demand Broker: PATCH instance (upgrade) On Demand Broker->>Service Adapter: generate-manifest Service Adapter->>On Demand Broker: manifest On Demand Broker->>BOSH: deploy BOSH->>On Demand Broker: accepted On Demand Broker->>Upgrade Errand: accepted Note over Upgrade Errand,BOSH: Upgrade Errand polls On Demand Broker for last operation until complete end Upgrade Errand->>Operator:completed successfully

## Delete All Service Instances

ODB provides BOSH errand to delete all the instances managed by the broker.

```
sequenceDiagram
    Operator->>Delete Errand:bosh run errand delete-all-service-instances
    Delete Errand->>Cloud Controller:GET instances
    Cloud Controller->>Delete Errand:instances loop for all instances
    Delete Errand->>Cloud Controller:GET bindings
    Cloud Controller->>Delete Errand:bindings loop for all bindings
    Delete Errand->>Cloud Controller:cf unbind-service
    Cloud Controller->>On Demand Broker:DELETE binding
    On Demand Broker->>Service Adapter:delete-binding
    Service Adapter->>On Demand Broker:exit code 0
    On Demand Broker->>Cloud Controller:OK
    Cloud Controller->>Delete Errand:OK end
    Delete Errand->>Cloud Controller:GET service keys
    Cloud Controller->>Delete Errand:service keys loop for all service keys
    Delete Errand->>Cloud Controller:cf delete-service-key
    Cloud Controller->>On Demand Broker:DELETE binding
    On Demand Broker->>Service Adapter:delete-binding
    Service Adapter->>On Demand Broker:exit code 0
    On Demand Broker->>Cloud Controller:OK
    Cloud Controller->>Delete Errand:OK end
    Delete Errand->>Cloud Controller:cf delete-service
    Cloud Controller->>On Demand Broker:DELETE instance
    On Demand Broker->>BOSH:delete deployment
    BOSH->>On Demand Broker:accepted
    On Demand Broker->>Delete Errand:accepted loop until DELETE completes
    Delete Errand->>Cloud Controller:GET instance
    Cloud Controller->>Delete Errand:delete instance in progress
    Note over Cloud Controller,BOSH: Cloud Controller asynchronously polls On Demand Broker, which in turn polls BOSH for last operation status
    Delete Errand->>Cloud Controller:GET instance
    Cloud Controller->>Delete Errand:instance not found end
    Delete Errand->>Operator:completed successfully
```

## Delete All Service Instances And Deregister Broker

ODB provides BOSH errand to delete all the instances managed by the broker and to deregister the broker from Cloud Foundry.

```
sequenceDiagram
    Operator->>Delete Errand:bosh run errand delete-all-service-instances
    Delete Errand->>Cloud Controller:POST disable-service-access
    Cloud Controller->>Delete Errand:completed successfully
    Delete Errand->>Cloud Controller:GET instances
    Cloud Controller->>Delete Errand:instances loop for all instances
    Delete Errand->>Cloud Controller:GET bindings
    Cloud Controller->>Delete Errand:bindings loop for all bindings
    Delete Errand->>Cloud Controller:cf unbind-service
    Cloud Controller->>On Demand Broker:DELETE binding
    On Demand Broker->>Service Adapter:delete-binding
    Service Adapter->>On Demand Broker:exit code 0
    On Demand Broker->>Cloud Controller:OK
    Cloud Controller->>Delete Errand:OK end
    Delete Errand->>Cloud Controller:GET service keys
    Cloud Controller->>Delete Errand:service keys loop for all service keys
    Delete Errand->>Cloud Controller:cf delete-service-key
    Cloud Controller->>On Demand Broker:DELETE binding
    On Demand Broker->>Service Adapter:delete-binding
    Service Adapter->>On Demand Broker:exit code 0
    On Demand Broker->>Cloud Controller:OK
    Cloud Controller->>Delete Errand:OK end
    Delete Errand->>Cloud Controller:cf delete-service
    Cloud Controller->>On Demand Broker:DELETE instance
    On Demand Broker->>BOSH:delete deployment
    BOSH->>On Demand Broker:accepted
    On Demand Broker->>Cloud Controller:accepted
    Cloud Controller->>Delete Errand:accepted loop until DELETE completes
    Delete Errand->>Cloud Controller:GET instance
    Cloud Controller->>Delete Errand:delete instance in progress
    Note over Cloud Controller,BOSH: Cloud Controller asynchronously polls On Demand Broker, which in turn polls BOSH for last operation status
    Delete Errand->>Cloud Controller:GET instance
    Cloud Controller->>Delete Errand:instance not found end
    Delete Errand->>Cloud Controller:DELETE service-broker
    Cloud Controller->>Delete Errand:completed successfully
    Delete Errand->>Operator:completed successfully
```

## Frequently asked questions

### How many dedicated service instances has Pivotal managed in a PCF environment?

The on-demand broker has been tested with 500 dedicated service instances using the [example Kafka on-demand tile](#).

We recorded how long it took to create, upgrade all and delete all, with 50, 101 and 500 dedicated service instances.

### Set up

Environment	
IaaS	Google Cloud Platform
PCF Operations Manager	v1.9.7
PCF Elastic Runtime	v1.9.13
Example Kafka On-Demand Tile	v0.15.1

BOSH Director Configuration	
Workers	3
Dedicated status worker	enabled

On-demand plan configuration	
Zookeeper VM type	small (1 CPU, 2GB RAM, 8GB Disk)
Kafka VM type	small (1 CPU, 2GB RAM, 8GB Disk)

### Test

1. Upload the example Kafka on-demand tile
2. Configure the on-demand plan
3. Apply changes to install the on-demand service, ensuring that `Register on-demand broker` is checked
4. Create N dedicated service instances using the CF CLI
5. Make a change to the plan configuration
6. Apply pending changes, ensuring that `Upgrade all on-demand service instances` is checked
7. Delete the tile and apply changes, ensuring that `Delete all on-demand service instances` is checked

### Results

Durations presented in HH:MM:SS format.

Create	50	101	500
average create	00:01:02	00:01:03	00:01:02
total	00:51:28	01:45:40	08:33:37

Upgrade All	50	101	500
average upgrade	00:01:10	00:01:05	00:01:00
total	00:58:37	01:49:42	08:21:08

Delete All	50	101	500
average delete	00:05:09	00:05:04	0:05:00
total	04:17:38	08:31:10	41:38:26

These durations may vary for a number of reasons, for example:

- Number of BOSH director workers
- IaaS performance
- Network latency
- Service instance BOSH release(s)
- Service instance deployment configuration
- VM type of service instance
- Activity of Elastic Runtime
- Activity of BOSH Director

## Notes

For create operations, the on-demand broker creates a BOSH deployment for each service instance. By default, the BOSH Director in Operations Manager v1.9 has three workers with a dedicated status worker, so only two workers are available to process deployment tasks. Therefore, only two service instances can be created at the same time.

For upgrade all and delete all operations, Operations Manager runs a BOSH errand. This errand task occupies a BOSH Director worker, leaving one worker available to upgrade, or delete deployments.