ICON R2 Application Build Book

Bluemix Deployment Instructions

Version 0.1

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 2017-03-08 | 0.1 | First draft | Stuart Thompson |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Components 4

2. Build Compose Services 5

2.1 Services Overview 5

2.2 Build icon-postgresql Service 6

2.2.1 Build procedure 6

2.3 Build icon-rabbitmq Service 6

2.3.1 Build procedure 6

2.4 Build icon-elasticsearch Service 7

2.4.1 Build procedure 7

3. Build IBM Containers 9

3.1 Container Overview 9

3.2 Build icon-clamav Container 9

3.2.1 Local Workstation Setup 9

3.2.2 Build ClamAv Container 9

3.2.3 Configure Public Access 11

4. Build Application Containers 12

4.1 Environment Setup 12

This procedure expects that the Compose services have been created and are running. This is important for the application deployment to be able to bind to the services. 12

A custom domain iconr2.marc-hi.ca is used to establish a branded name, which is different from the default bluemix.net domain. 12

4.1.1 Domain Name Setup 12

4.1.2 Local Workstation Setup 12

4.2 Build dhir-nginx Container 12

4.3 Build icon-server and icon-server-worker Containers 13

4.4 Build icon-nginx Container 13

4.5 Build icon-kibana Container 14

4.6 Build the Database 14

5. Operational Procedures 15

5.1 Alter Endpoint URL 15

# Introduction

This document is a set of instructions to deploy the Immunize Connect Ontario (ICON) application to the Bluemix hosting environment. The ICON application lets public users submit and retrieve immunization records over the internet.

## Purpose

The ICON application is hosted by the IBM Bluemix Dedicated cloud computing platform, from within an IBM SoftLayer data center. IBM Bluemix is a platform as-a-service (PaaS) for computing, storage and networking capacity. ICON is built upon technologies that are supported by Bluemix services.

The deployment and operation of the ICON application is largely based on the configuration of the IBM Bluemix environment, which is described in a related document *Physical Deployment Document*.

## Components

The components of the ICON application are shown in Figure 1.2. They are a combination of Compose Services, IBM Containers and Cloud Foundry Containers. The figure shows the interaction dependencies between the application components.

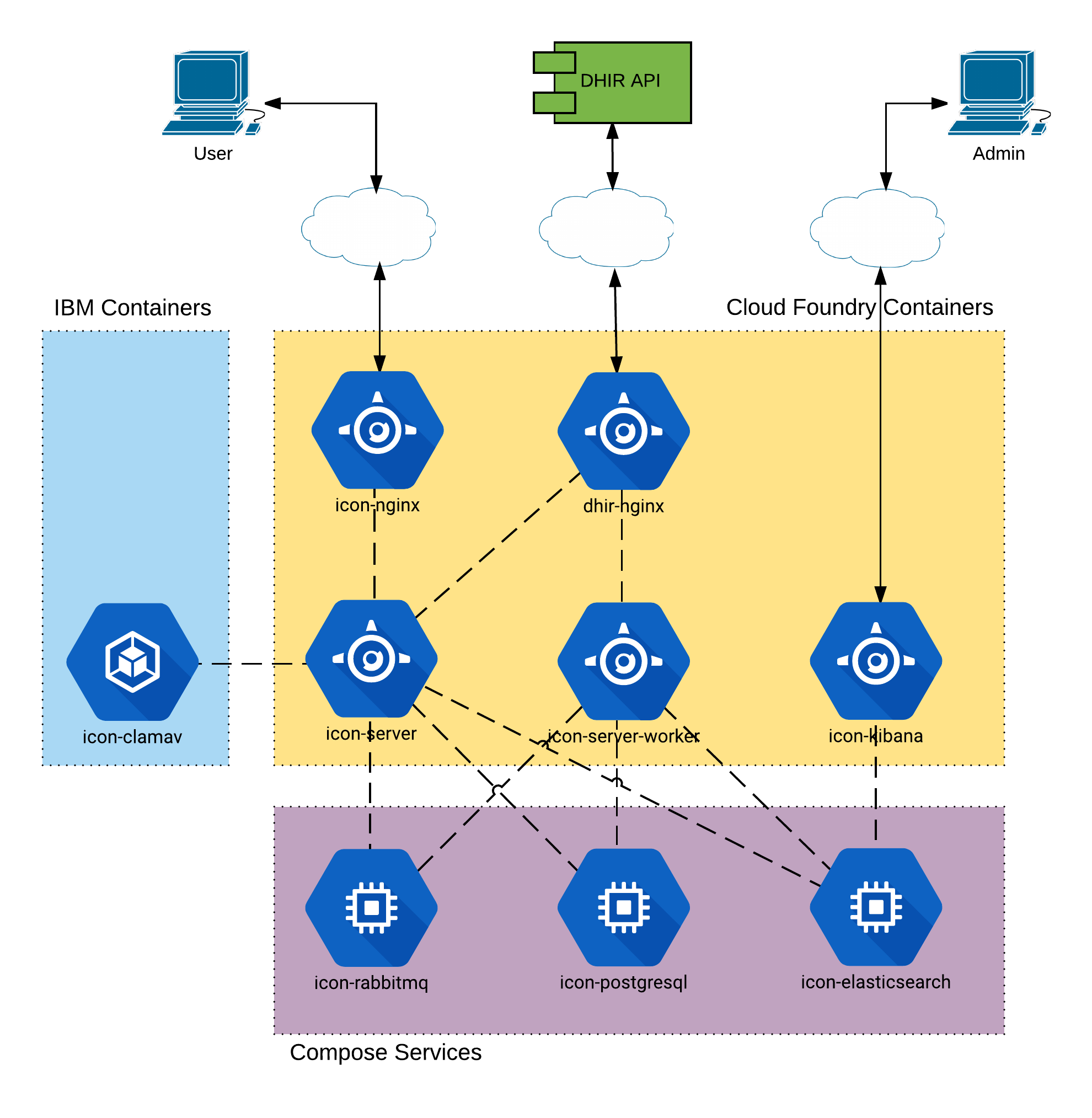


Figure 1.2: Component Dependencies

# Build Compose Services

## Services Overview

The ICON project relies on Bluemix Compose services. A Compose service is typically a highly-available three-node cluster of virtual machines which run a specific product. It is equipped with a front-facing load balancer to deal with failover scenarios. To the consuming application, the administrative details of the cluster are hidden.

Those Compose services employed by the ICON application are summarized by the following table.

|  |  |  |
| --- | --- | --- |
| Service Name | | Description |
| icon-postgresql | Highly-available PostgreSQL relational database | |
| icon-rabbitmq | Highly-available RabbitMQ queuing product | |
| icon-elasticsearch | Highly-available Elasticsearch for log storage and query | |

To provision Compose services the Bluemix Console is used. Figure 2.1.1 shows the Bluemix catalog view for the menu Services > Data & Analytics.

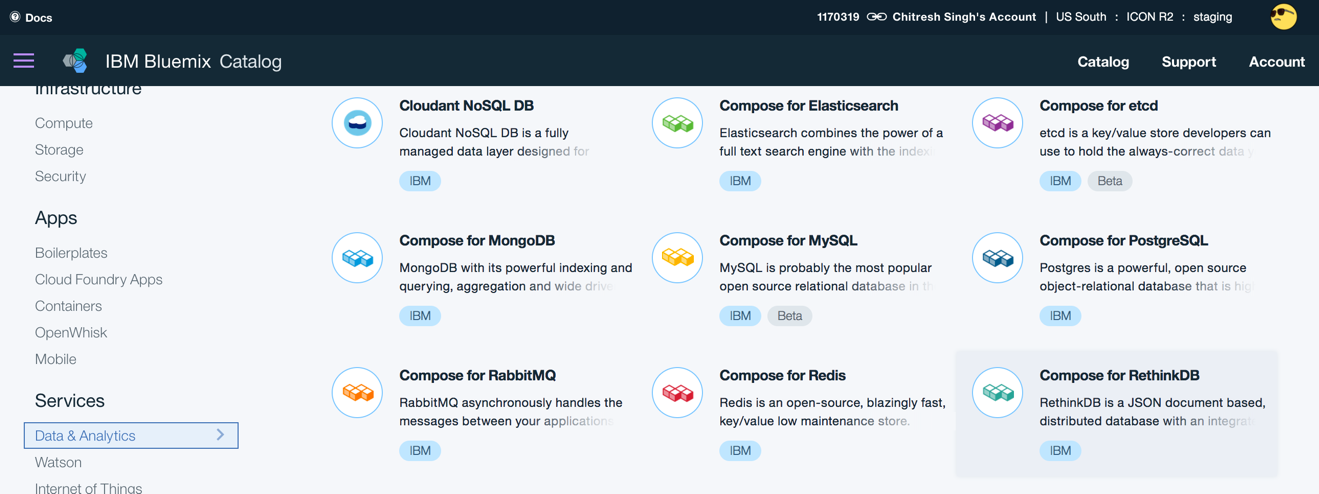


Figure 2.1.1: Bluemix Compose Services.

When provisioning a Compose service in Bluemix Public, the service name is specific to each Space within an Organization. This allows the same service name to be used without conflict between spaces. For example, development and staging environments can reuse the same service name.

The service credentials for each Compose service is automatically generated during service creation. Using the credential, an application can connect to the service; alternatively, the service can be connected to using external tools. From the Bluemix Console, click on the icon for the specific service, then menu Service Credentials, where the credentials can be viewed in JSON format.

## Build icon-postgresql Service

This is a manual procedure, which is performed only for the initial installation of the product. The default configuration is 1GB storage, 1GB RAM which automatically scales with the data.

### Build procedure

Step 1: As shown in Figure 2.2.1, from the Bluemix Console, select Category > Services > Data & Analytics.

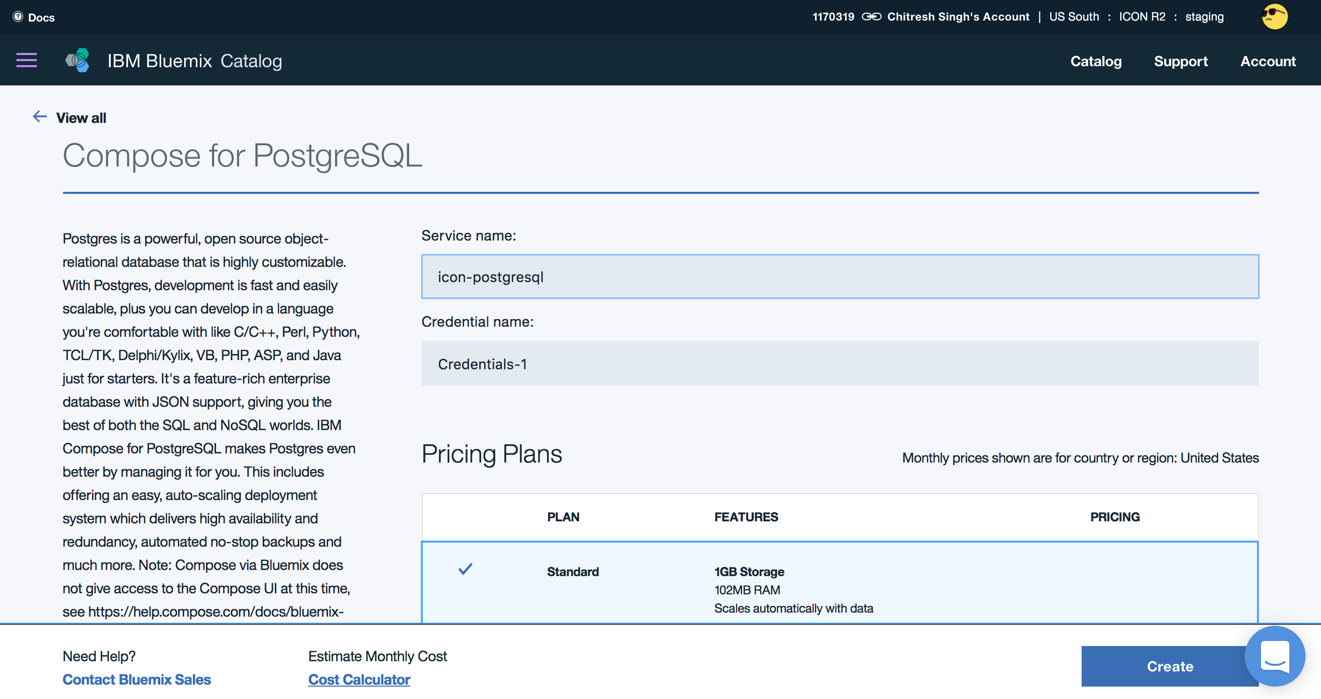


Figure 2.2.1: Compose for PostgreSQL

Step 2: Click on the icon for *Compose for PostgreSQL.*

Step 3: Enter Service name: icon-postgresql.

Step 4: Leave the default Credentials name. Strong credentials are randomly generated then programmatically retrieved through the service-to-application binding. Binding is automatically done during deployment when listed in an application’s manifest file.

Step 5: Click the Create button. The highly-available instance will be created within a few minutes. When the service is created, it appears in the Services section of the Bluemix Console.

## Build icon-rabbitmq Service

This is a manual procedure, which is performed only for the initial installation of the product. The default configuration is 256MB storage.

### Build procedure

Step 1: As shown in Figure 2.3.1, from the Bluemix Console, select Category > Services > Data & Analytics.

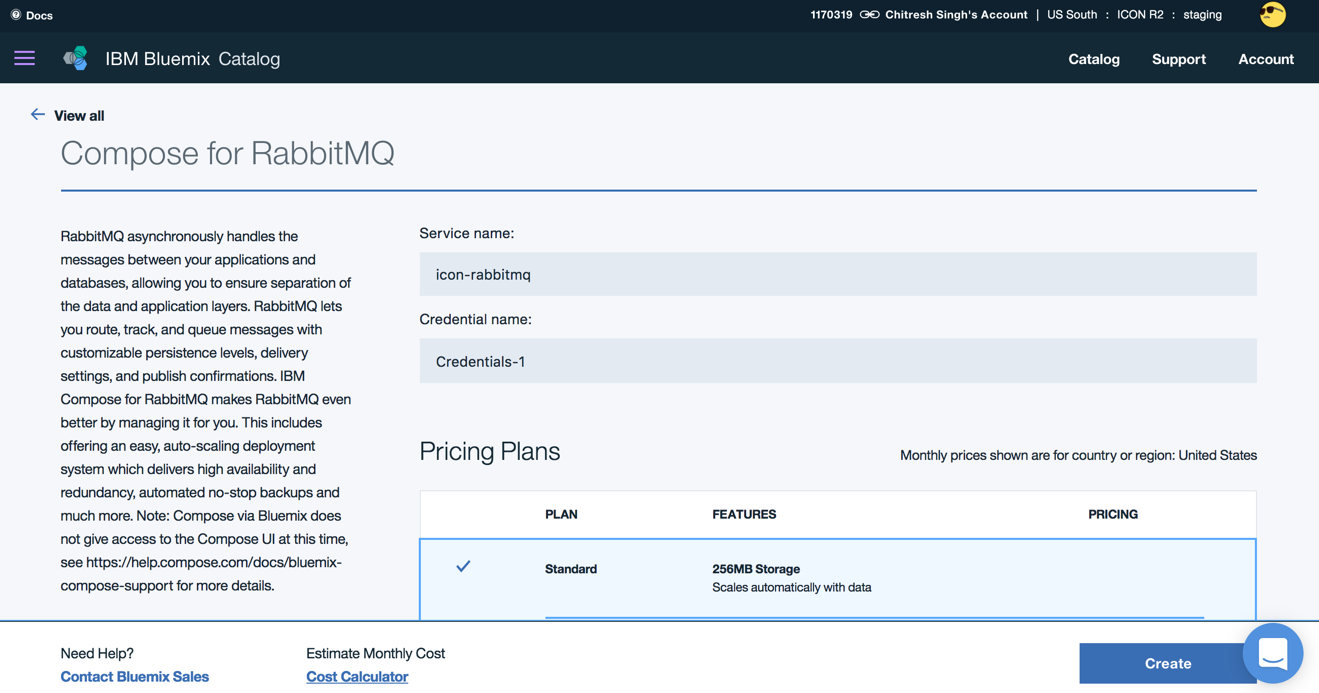


Figure 2.3.1: Compose for RabbitMQ

Step 2: Click on the icon for *Compose for RabbitMQ.*

Step 3: Enter Service name: icon-rabbitmq.

Step 4: Leave the default Credentials name. Strong credentials are randomly generated then programmatically retrieved through the service-to-application binding. Binding is automatically done during deployment when listed in an application’s manifest file.

Step 5: Click the Create button. The highly-available instance will be created within a few minutes. When the service is created, it appears in the Services section of the Bluemix Console.

## Build icon-elasticsearch Service

This is a manual procedure, which is performed only for the initial installation of the product. The default configuration is 2GB storage, 2GB RAM which automatically scales with the data.

### Build procedure

Step 1: As shown in Figure 2.4, from the Bluemix Console, select Category > Services > Data & Analytics.

Step 2: Click on the icon for *Compose for Elasticsearch.*

Step 3: Enter Service name: icon-elasticsearch.

Step 4: Leave the default Credentials name. Strong credentials are randomly generated then programmatically retrieved through the service-to-application binding. Binding is automatically done during deployment when listed in an application’s manifest file.

Step 5: Click the Create button. The highly-available instance will be created within a few minutes. When the service is created, it appears in the Services section of the Bluemix Console.

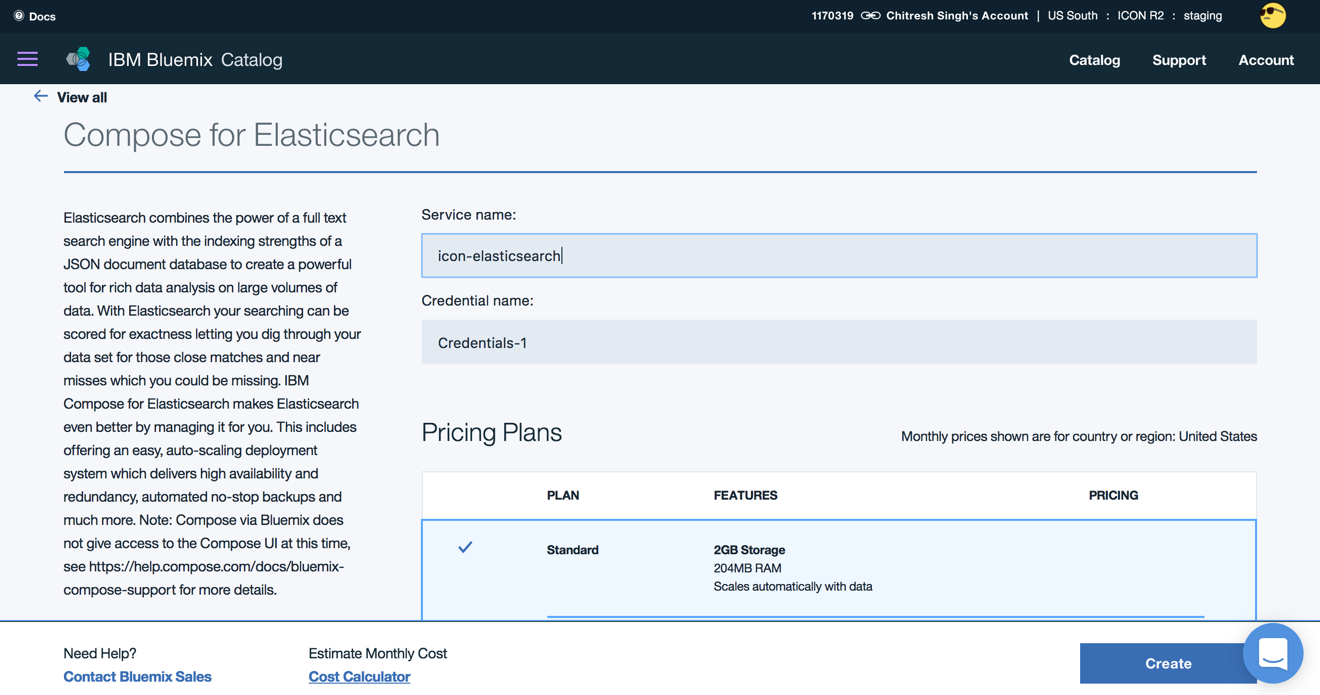


Figure 2.4: Compose for Elasticsearch

# Build IBM Containers

## Container Overview

The ICON project relies on Docker containers, which operate within the Bluemix IBM Container service. The Docker container is virtual software that includes all of the operating system dependencies to run an application, which is built into an image. Once a Docker image is deployed to Bluemix, an instance container of the Docker image can be instantiated.

## Build icon-clamav Container

ICON uses the ClamAv product to scan for viruses within the file attachments that a user supplies as part of the immunization submission. ClamAv runs as a Docker container that is retrieved from the official Docker repository.

For Bluemix Public, the name of an IBM Container image is global to the organization and visible in all spaces within an organization. On the other hand, the name of an IBM Container instance is specific to each space within an organization.

NOTE: Some of these configuration settings will be different for the Bluemix Dedicated environment, but at the time of this document these are not known.

### Local Workstation Setup

Step 1: Install the cloud foundry command-line interface (CLI) on the workstation that will doing the deployment from <https://github.com/cloudfoundry/cli/releases>.

Step 2: The region of the API endpoint is specified. Example shown for Public US South region.

$ cf api https://api.ng.bluemix.net

Step 3: Login to Bluemix container service “staging” space, where <user> is a Bluemix account name.

$ cf login –u <user> -o “ICON R2” –s staging

Step 4: Install the IBM Container command-line interface (IBM CLI). Example shown for Mac.

$ cf install-plugin https://static-ice.ng.bluemix.net/ibm-containers-mac

Step 5: The first time the IBM CLI must be initialized. Login to the IBM Container subsystem.

$ cf ic init

$ cf ic login

Step 6: Install the Docker engine from site <https://docs.docker.com/engine/installation>. Follow the procedure appropriate to the operating system (e.g. Mac, Windows, Linux).

### Build ClamAv Container

Step 7: Build a local Docker image for ClamAv. This pulls the Debian image and virus database from the official Docker repository, which can take an hour to complete.

$ docker build –t icon/clamav .

Step 8: Tag and push the ClamAv Docker image to Bluemix. This step automatically invokes the Bluemix Vulnerability Advisor to run a virus scan on the Docker image. Until the virus scan is complete, the image cannot be used.

$ docker tag icon/clamav registry.ng.bluemix.net/icon/clamav

$ docker push registry.ng.bluemix.net/icon/clamav

Step 9a: View the list of Docker images to verify. Figure 3.2.2a shows the Bluemix Console from the menu Category > Apps > Containers, where the clamav container is visible.

$ cf ic images

Step 9b: Click on the “clamav” icon to the see the container detail page. When the virus scan is complete, the status is shown under the label “Vulnerability Assessment”. This is shown in Figure 3.2.2b.

$ cf ic images

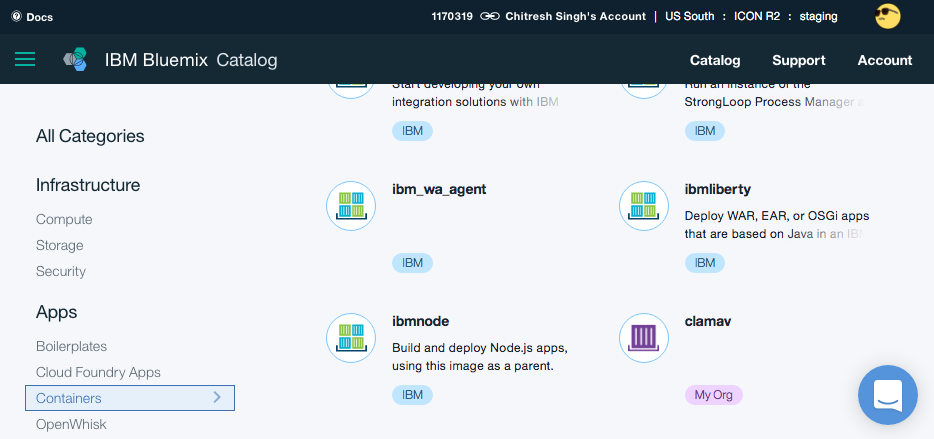


Figure 3.2.2a: View IBM Containers

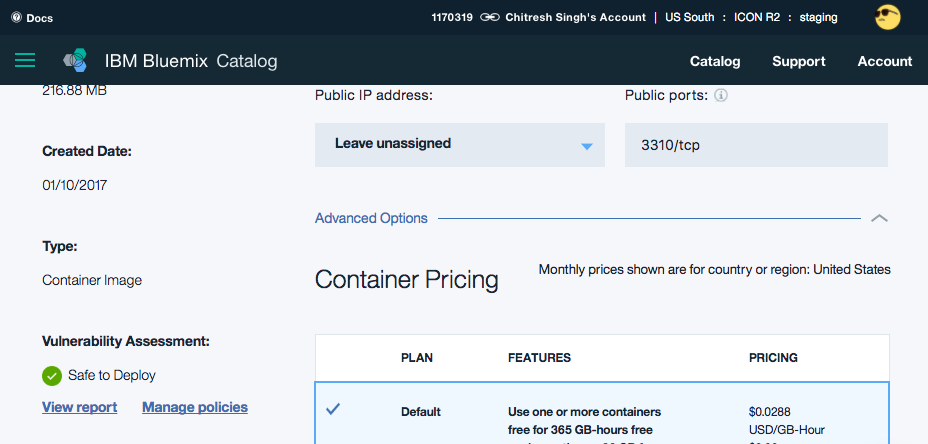


Figure 3.2.2b: Virus Scan Status

Step 10a: Create an IBM Container instance of ClamAv, named “icon-clamav” and set its service port.

$ cf ic run –p 3310:3310 -–name icon-clamav

registry.ng.bluemix.net/icon/clamav

Step 10b: Bluemix automatically assigns a private IP address to the container. Obtain the IP address of *icon-clamav* container, so it can be set to *icon-server* configuration. For example, 172.31.0.3.

$ cf ic inspect --format "{{ .NetworkSettings.IPAddress }}" icon-clamav

### Configure Public Access

These steps are only required for development environments, where there is a need for icon-clamav to be accessed from outside the Bluemix environment. For this purpose, an IBM Container requires a public IP address be allocated to the space.

Step 11a: Check if any IP addresses are available.

$ cf ic ip list

Step 11b: If no IP addresses are listed, request Bluemix to allocate a public IP address. Use that address in the next step.

$ cf ic ip request

Step 11c: Bind the public IP address to the *icon-clamav* container.

$ cf ic ip bind 169.44.125.163 icon-clamav

# Build Application Containers

The majority of components of the ICON project consists of Cloud Foundry application containers. Those containers employed by the ICON application are summarized by the following table.

|  |  |  |
| --- | --- | --- |
| Container Name | | Description |
| dhir-nginx | NGINX buildpack for outbound communication to DHIR | |
| icon-server | Node.js buildpack to handle requests from ICON UI | |
| icon-server-worker | Node.js buildpack for async. processing of the ICON queue | |
| icon-nginx | NGINX buildpack packaged with the ICON UI | |
| icon-kibana | Node.js buildpack instance of Kibana for Bluemix | |

## Environment Setup

### This procedure expects that the Compose services have been created and are running. This is important for the application deployment to be able to bind to the services.

### A custom domain iconr2.marc-hi.ca is used to establish a branded name, which is different from the default bluemix.net domain.

### Domain Name Setup

Step 1a: Create two custom domains for the organization in the “staging” space: “stg” (front-end) and “stg-backend” (back-end). The back-end is protected from external access.

$ cf create-domain “ICON R2” stg.iconr2.marc-hi.ca

$ cf create-domain “ICON R2” stg-backend.iconr2.marc-hi.ca

Step 1b: List the custom domains.

$ cf domains

### Local Workstation Setup

Step 2: Install the cloud foundry command-line interface (CLI) on the workstation that will doing the deployment from <https://github.com/cloudfoundry/cli/releases>.

Step 3: The region of the API endpoint is specified. Example shown for Public US South region.

$ cf api https://api.ng.bluemix.net

Step 4: Login to Bluemix container service “staging” space, where <user> is a Bluemix account name.

$ cf login –u <user> -o “ICON R2” –s staging

## Build dhir-nginx Container

The NGINX outbound container is configured for 512MB disk, 512MB RAM and the *Staticfile* buildpack.

The *nginx.conf* configuration file is located in the *public* directory. Important configuration parameters are:

* proxy\_pass - the DHIR endpoint URL
* proxy\_ssl\_certificate – path to certificates (stored in the *ssl* directory)

The *stg-manifest.yml* is located in the *bluemix* directory. It specifies the domain “stg-backend.iconr2.marc-hi.ca”, which results in the deployed container route “dhir-nginx.stg-backend.iconr2.marc-hi.ca”.

Step 1: Navigate to the *dhir-nginx* directory containing the application.

$ cd dhir-nginx

Step 2: Push the build for the “staging” environment.

$ cf push –f bluemix/stg-manifest.yml

## Build icon-server and icon-server-worker Containers

The ICON Server consists of two containers names “icon-server” and “icon-server-worker”, which are each configured for 1GB disk, 2GB RAM and the Node.js buildpack.

The *stg-manifest.yml* is located in the *bluemix* directory. It specifies the domain “stg-backend.iconr2.marc-hi.ca”, which results in the deployed container route “icon-server.stg-backend.iconr2.marc-hi.ca”, but configured that no route is created for the “icon-server-worker” container (to keep it private).

Step 1: Navigate to the *backend-v2* directory containing the application.

$ cd backend-v2

Step 2: Navigate to the *bluemix* directory containing the build configuration.

$ cd bluemix

Step 3: Edit the *stg-manifest.yml* file to set the following environment variables.

CLAMAV\_ENDPOINT: ‘172.31.0.3’

PHIX\_ENDPOINT\_SUBMISSION: ‘https://dhir-nginx.stg-backend.iconr2.marc-hi.ca/Communication’

PHIX\_ENDPOINT\_RETRIEVAL: ‘https:// dhir-nginx.stg-backend.iconr2.marc-hi.ca /Immunization’

Option: Development simulator environment variables.

CLAMAV\_ENDPOINT: ‘172.31.0.3’

PHIX\_ENDPOINT\_SUBMISSION: ‘https://panorama-simulator.mybluemix.net/Communication’

PHIX\_ENDPOINT\_RETRIEVAL: ‘https://panorama-simulator.mybluemix.net/Immunization’

Step 4: Push the build for the respective environment.

$ cf push –f stg-manifest.yml

Option: To push only one of the two applications, specify the application name.

$ cf push icon-server-worker –f stg-manifest.yml

## Build icon-nginx Container

The NGINX inbound container is configured for 512MB disk, 512MB RAM and the *Staticfile* buildpack.

The *nginx.conf* configuration file is located in the *main* directory. If any changes are made to its configuration, the application distribution must be rebuilt using “npm start”.

The *stg-manifest.yml* is located in the *bluemix* directory. It specifies the domain “stg.iconr2.marc-hi.ca”, which results in the deployed container route “icon-nginx.stg.iconr2.marc-hi.ca”.

Step 1: Navigate to the *frontend-v2* directory containing the application.

$ cd frontend-v2

Step 1: Build the application distribution, which outputs to the *dist* directory.

$ npm start

Step 2: Push the build for the “staging” environment.

$ cf push –f bluemix/stg-manifest.yml

## Build icon-kibana Container

The Kibana application is configured for 1GB disk, 2GB RAM and the Node.js buildpack

The *stg-manifest.yml* is located in the *bluemix* directory. It specifies the domain “stg-backend.iconr2.marc-hi.ca”, which results in the deployed container route “dhir-nginx.stg-backend.iconr2.marc-hi.ca”.

Step 1: Navigate to the *icon-kibana* directory containing the application.

$ cd icon-kibana

Step 2: Push the build for the “staging” environment.

$ cf push –f bluemix/stg-manifest.yml

## Build the Database

Step 1: Navigate to the *postgres-v2* directory containing the database scripts.

$ cd postgre-v2

Step 2: Run the SQL scripts to create the database.

$ psql –h <hostname> -p <port> -d compose –U admin –a –f 1\_Add\_Roles.sql

$ psql –h <hostname> -p <port> -d compose –U admin –a –f 1\_Create\_Lookup.sql

$ psql –h <hostname> -p <port> -d compose –U admin –a –f 1\_Create\_Submit.sql

Step 3: Run the SQL scripts to populate the database.

$ tar xvf 4\_Geocode\_Data.sql.zip

$ psql –h <hostname> -p <port> -d compose –U admin –f 4\_Geocode\_Data.sql

$ psql –h <hostname> -p <port> -d compose –U admin –a –f 5\_Lookup\_Data.sql

# Operational Procedures

## Alter Endpoint URL

It is possible to change the DHIR endpoint URL without a deployment, by setting the environment variables of the *icon-server* and *icon-server-worker* containers.

Some relevant environment variables are summarized by the following table. Figure 5.1 shows the page for the *icon-server-worker* container.

|  |  |  |
| --- | --- | --- |
| Variable Name | | Description of Value |
| PHIX\_ENDPOINT\_SUBMISSION | DHIR submission endpoint URL | |
| PHIX\_ENDPOINT\_RETRIEVAL | DHIR retrieval endpoint URL | |
| NODE\_TLS\_REJECT\_UNAUTHORIZED | Set to 0 if using self-signed certificate | |

Step 1: From the Bluemix Console, select the *icon-server* container.

Step 2: Select Runtime in the left menu, then scroll down to the User Defined section. Click the Add button, and enter the name and value for the environment variable.

Step 3: Repeat for each environment variable. Click the Save button, to restart the container.

Step 4: Repeat the procedure for the *icon-server-worker* container.

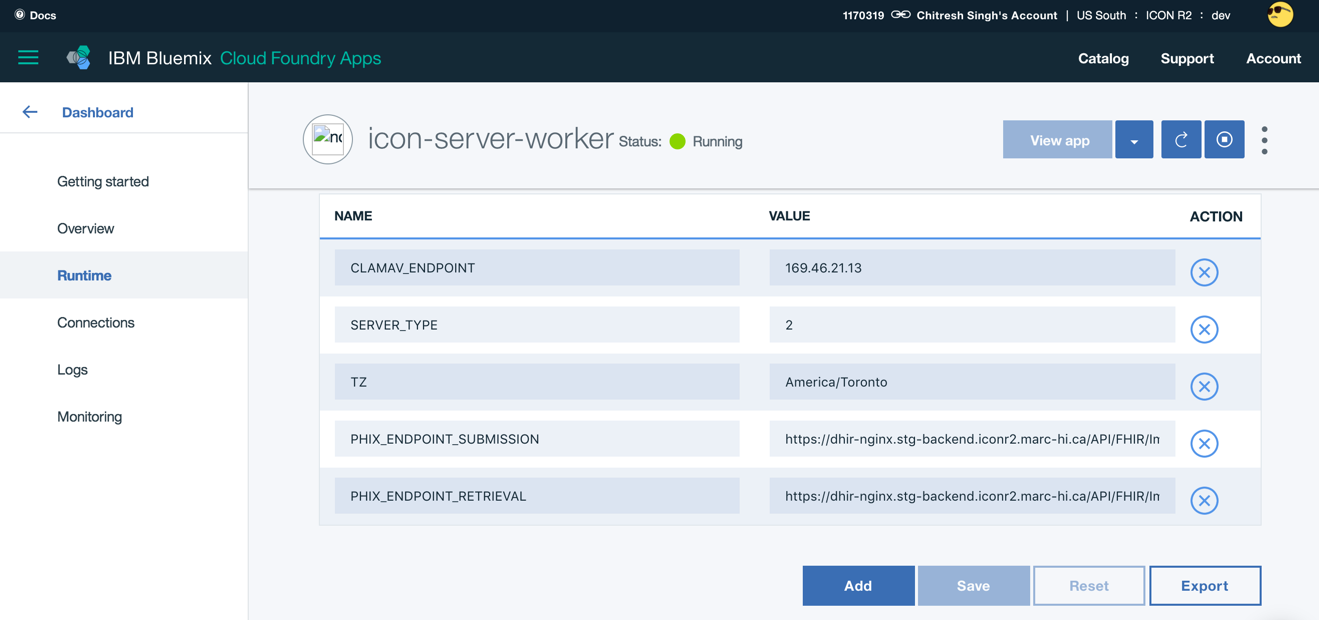


Figure 5.1: Container environment variables.