Predix Message Queue

Contents

Copyright GE Digital

Predix Message Queue Overview

About Predix Message Queue

Predix Message Queue, based on RabbitMQ Message Broker, is a service that implements the AMQP protocol.

Predix Message Queue, based on the RabbitMQ message broker software provides persistent, highly available, reliable messaging between applications, components, and devices. The message queue service supports a variety of messaging protocols, many clients, and flexible routing with built-in direct, fanout, topic, and header exchange types.

Predix Message Queue Subscription Plans

Choose the subscription plan best suited to your environment and needs.

Predix Message Queue Dedicated Single-Node Configuration

Feature	Description
High Availability	None; the service will be interrupted upon a host reboot.
Backups	Hourly with 24-hour retention.
Dedicated VM	Single RabbitMQ process.
Application	Advanced development and testing requirements; Production environment where highly available configuration is not required.
Queue Policies	Each queue can hold up to 5 GB of data, and will keep messages for 30 days.
vCPU	4
Storage	80 GB

Predix Message Queue Dedicated and Highly Available Configuration

Feature	Description
High Availability	Three-node RabbitMQ cluster configured for single tenant use will recover automatically form a failure of the master with no impact on reads and minimal impact on writes. No single point of failure.
Persistence	User definable Durable Queue settings
Backups	Hourly with 24-hour retention
Application	Production environments that require a Highly Available configuration.
Queue Policies	Each queue can hold up to 5 GB of data, and will keep messages for 30 days.
vCPU	4 per node
Storage	400 GB

Getting Started with Predix Message Queue

Creating a Predix Message Queue Service Instance

About This Task

You can use Cloud Foundry commands to view available service plans in the Cloud Foundry marketplace.

Procedure

1. Use the `cf marketplace` command to view the available services. For example,

```
$ cf marketplace
Getting services from marketplace in org predix-services / space ref-
apps as adam.duston@ge.com...
OK

service
plans
        description
predix-message-queue     Dedicated-1-Q20*, Dedicated-3HA-Q20*
```

2. Use the `cf create-service` command to create a new instance of predix-message-queue. Specify the desired plan and a name for the new service instance. Example:

```
$ cf create-service predix-message-queue Dedicated-1-Q20 my-predix-
message-queue
Creating service instance my-predix-message-queue in org predix-
services / space ref-apps as adam.duston@ge.com...
OK

Create in progress. Use 'cf services' or 'cf service my-predix-
message-queue' to check operation status.

Attention: The plan `Dedicated-1-Q20` of service `predix-message-
queue` is not free.
The instance `my-predix-message-queue` will incur a cost.  Contact
your administrator if you think this is in error.
```

3. Once the service instance is created, you should receive a status of `create succeeded`. You can then bind the service instance to an app with the `cf bind-service` command. Example:

```
cf bind-service my-app my-predix-message-queue
```

Retrieving Predix Message Queue Environment Variables

About This Task

Cloud Foundry will expose information about the Predix Message Queue service bound to the application in JSON format using an environment variable called `VCAP_SERVICES`. This should contain everything that you need to know to interact with service instance. You can view the contents of the `VCAP_SERVICES` variable for a given app using the cf env command. Here's an example:

Procedure

Example of `VCAP_SERVICES`

```
{
 "VCAP_SERVICES": {
  "predix-message-queue": [
   {
    "credentials": {
     "dashboard_url": "https://rabbitmq-8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d.run.aws-usw02-sb.ice.predix.io/#/login/
8b12ecb4-0edc-451b-8c1d-d9c525b42a9d/Dm9eDcpWasaGOtHWESkCXg",
     "hostname": "10.72.74.45",
     "hostnames": [
      "10.72.74.45"
     ],
     "http_api_uri": "https://8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d:Dm9eDcpWasaGOtHWESkCXg@rabbitmq-8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d.run.aws-usw02-sb.ice.predix.io/api/",
     "http_api_uris": [
      "https://8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d:Dm9eDcpWasaGOtHWESkCXg@rabbitmq-8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d.run.aws-usw02-sb.ice.predix.io/api/"
     ],
     "password": "Dm9eDcpWasaGOtHWESkCXg",
     "protocols": {
      "amqp": {
       "host": "10.72.74.45",
       "hosts": [
        "10.72.74.45"
       ],
       "password": "Dm9eDcpWasaGOtHWESkCXg",
       "port": 5672,
       "ssl": false,
       "uri": "amqp://8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d:Dm9eDcpWasaGOtHWESkCXg@10.72.74.45:5672/8b12ecb4-0edc-451b
-8c1d-d9c525b42a9d",
       "uris": [
        "amqp://8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d:Dm9eDcpWasaGOtHWESkCXg@10.72.74.45:5672"
       ],
       "username": "8b12ecb4-0edc-451b-8c1d-d9c525b42a9d",
       "vhost": "8b12ecb4-0edc-451b-8c1d-d9c525b42a9d"
      },
      "management": {
       "host": "10.72.74.45",
       "hosts": [
        "10.72.74.45"
       ],
       "password": "Dm9eDcpWasaGOtHWESkCXg",
       "path": "/api/",
       "port": 15672,
       "ssl": false,
       "uri": "http://8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d:Dm9eDcpWasaGOtHWESkCXg@10.72.74.45:15672/api/",
       "uris": [
        "http://8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d:Dm9eDcpWasaGOtHWESkCXg@10.72.74.45:15672"
       ],
       "username": "8b12ecb4-0edc-451b-8c1d-d9c525b42a9d",
       "vhost": "8b12ecb4-0edc-451b-8c1d-d9c525b42a9d"
```

```
      }
     },
     "ssl": false,
     "uri": "amqp://8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d:Dm9eDcpWasaGOtHWESkCXg@10.72.74.45/8b12ecb4-0edc-451b-8c1d
-d9c525b42a9d",
     "uris": [
      "amqp://8b12ecb4-0edc-451b-8c1d-
d9c525b42a9d:Dm9eDcpWasaGOtHWESkCXg@10.72.74.45/8b12ecb4-0edc-451b-8c1d
-d9c525b42a9d"
     ],
     "username": "8b12ecb4-0edc-451b-8c1d-d9c525b42a9d",
     "vhost": "8b12ecb4-0edc-451b-8c1d-d9c525b42a9d"
    },
    "label": "predix-message-queue",
    "name": "my-predix-message-queue",
    "plan": "Dedicated-1-Q20",
    "provider": null,
    "syslog_drain_url": null,
    "tags": [
     "rabbitmq"
    ],
    "volume_mounts": []
   }
  ]
 }
}
```


The `predix-message-queue` key at the root of the `VCAP_SERVICES` json should contain a list of all bound Predix Message Queue services for the app. Within that key is all of the information necessary to interact with the service.

VCAP Services Key-Value Pair Definitions

VCAP Key-Value Pairs

This section describes each key in the `VCAP_SERVICES` JSON for the Predix Message Queue service and its purpose.

VCAP Key	Value Definition
label:	The name of the Cloud Foundry service this instance belongs to, in this case predix-message-queue.
name:	The name of the service instance.
plan:	The service plan for the instance.
provider:	Not used for this service.
syslog_drain_url:	Syslog URL for logging services. Not used for RabbitMQ.
tags:	List of tags about the type of service. Can be used for identifying the type of an attached service.
volume_mounts:	Metadata about attached storage volumes. Not used for this service.

Credentials Keys

The `credentials` key contains the information necessary to connect and authenticate to the Predix Message Queue service.

Credential Key	Value Definition
`dashboard_url :`	The URL to the RabbitMQ web management interface.
`hostname :`	The hostname or IP address of the primary RabbitMQ service node.
`hostnames:`	A list of hostnames or URLs for all of the nodes in the RabbitMQ cluster.
`http_api_uri:`	The URI for the RabbitMQ HTTP API.
`http_api_uris:`	A list of URIs for the HTTP management API.
`password:`	The password for the service's RabbitMQ user account

Protocols Keys

Protocols are specified as a subsection of Credentials key. The `protocol` keys provide all of the necessary details for connecting to the service with various protocols

Protocol Key	Value Definition
`amqp :`	Credentials (host, username, password, vhost, uri etc) necessary for creating AMQP connections to the service.
`management :`	Credentials for the RabbitMQ management API.

Using Predix Message Queue

Code Examples for Use with Predix Message Queue

Guide to Using Code Examples with Predix Message Queue

The examples included here demonstrate how to use the Predix Message Queue service with each of our supported build pack languages.

These examples are based on the official RabbitMQ Tutorials. You can find more information about working with the RabbitMQ service there. These examples all demonstrate using direct routing with a routing key as shown in tutorial 4, Routing in the official tutorials.

Code examples are included for these languages:

- Java
- Ruby
- Node.js
- Python
- PHP
- GO

Java Code Examples for Predix Message Queue

Recommended Library

RabbitMQ Java Client

Alternative Libraries

Spring AMQP

Java Code Example using RabbitMQ Java Client

Include the RabbitMQ client as a Mavcen dependency:

```
<dependency>
  <groupId>com.rabbitmq</groupId>
  <artifactId>amqp-client</artifactId>
  <version>4.2.0</version>
</dependency>
```

Configure the connection to your Predix-Message-Queue service using the `VCAP_SERVICES`environment variable. In this example we're using the `JSON-java` library to parse the JSON data in`VCAP_SERVICES`.

```
// Parse the RabbitMQ config from the VCAP_SERVICES environment.
Assumes there's only one RabbitMQ instance bound.
String vcapServices = System.getenv("VCAP_SERVICES");
JSONObject vcapJson = new JSONObject(vcapServices);
JSONObject amqpCredentials =  vcapJson.getJSONArray("predix-message-
queue") //
        .getJSONObject(0) //
        .getJSONObject("credentials")
```

```
        .getJSONObject("protocols")
        .getJSONObject("amqp");

/* Set up the RabbitMQ connection. Alternatively you can also use the
URI:
*
*   factory.setUri(amqpCredentials.getString("uri"));
*/
ConnectionFactory factory = new ConnectionFactory();
factory.setHost(amqpCredentials.getString("host"));
factory.setUsername(amqpCredentials.getString("username"));
factory.setPassword(amqpCredentials.getString("password"));
factory.setPort(amqpCredentials.getInt("port"));
factory.setVirtualHost(amqpCredentials.getString("vhost"));
Connection connection = factory.newConnection();
```

Configure a `Channel` to declare a `Queue`. You can bind the `Queue` to an `Exchange` with a routing key as needed for your specific use case (See the RabbitMQ Tutorials for more details):

```
// Define Queue name, Exchange name, and routing key as strings.
String QUEUE_NAME = "PredixRabbitMQSample";
String EXCHANGE_NAME = "sample-exchange";
String ROUTING_KEY = "test";

// Create the channel, declare the queue and exchange, and bind them.
Channel channel = connection.createChannel();
channel.exchangeDeclare(EXCHANGE_NAME, "direct", true);
channel.queueDeclare(QUEUE_NAME, true, false, false, null);
channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, ROUTING_KEY);
```

The simplest way to publish messages is to use the the `basicPublish` method of the `Channel`. Convert the message data to bytes before publishing. In this example the message body is a randomly generated UUID string.

```
// Generate a random UUID to use for the message
String message =  UUID.randomUUID().toString();
byte[] messageBodyBytes = message.getBytes();

// Publish the message using basicPublish
channel.basicPublish(EXCHANGE_NAME, ROUTING_KEY, null,
messageBodyBytes);
```

Messages are consumed asynchronously. Messages are pushed from the queue to the consumer, so you need to define a callback method that will run when a message is delivered to your app. You can do this using the `Consumer` class. First, declare the `Queue`and create the `Channel` as described above, then configure the `Consumer`:

```
Consumer consumer = new DefaultConsumer(channel) {
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties basicProperties,
                               byte[] body) throws IOException {
        String message = new String(body, "UTF-8");
        String output = String.format("Received message from exchange
%s: %s", envelope.getExchange(), message);
        System.out.println(output);

    }
```

```
};
channel.basicConsume(queueName, true, consumer);
```

Spring AMQP Client Code Sample for Predix Message Queue

Spring Boot Framework Examples

The Spring Boot framework offers some libraries that simplify the process of integrating with services in CloudFoundry. Services will be auto-discovered without the need to manually parse the `VCAP_SERVICES` environment variable. Here's how to achieve results similar to the above example using Spring AMQP.

Include the spring-cloud-dependencies in your Maven configuration:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Dalston.SR1</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Also include the Spring AMQP library dependencies:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-amqp</artifactId>
        <version>1.5.4.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-bus-amqp</artifactId>
        <version>1.3.1.RELEASE</version>
    </dependency>
</dependencies>
```

To configure the Queue, Exchange, and Routing key names you can define a `Configuration`class which injects the following Beans:

```
@Configuration
public class RabbitMQSpringSampleConsumerConfig {

    @Bean
    public Queue queue() {
        return new Queue("PredixRabbitMQSpringSample");
    }

    @Bean
    public DirectExchange exchange() {
        return new DirectExchange("sample-exchange");
    }

    @Bean
```

```
    public Binding binding(DirectExchange exchange, Queue queue) {
        return BindingBuilder.bind(queue).to(exchange).with("test");
    }
}
```

Then, to publish messages to the Queue inject the following `Beans` in your producing app:

```
@Autowired
private DirectExchange exchange;

@Autowired
private RabbitTemplate template;
```

These will be configured with the names and binding details from the `Configuration` class when your app starts. You can now use them to publish messages to the queue:

```
// Generate a random UUID to use for the message
String message = UUID.randomUUID().toString();

// Publish the message to the configured exchange, using 'test' as the
routing key.
template.convertAndSend(exchange.getName(), "test", message);
```

Consuming messages is accomplished using the `RabbitListener` class. Use a similar Configuration class as above to configure the queue and bind it to an exchange, then set up the `RabbitListener` for that queue and define a callback method to handle incoming messages:

```
@RabbitListener(queues="#{queue.name}")
public void receiveMessage(String data) throws InterruptedException {
    String exchangeName = exchange.getName();
    System.out.println(String.format("Received message from exchange
%s: %s", exchangeName, data));
}
```

Ruby Code Sample for Predix Message Queue

Recommended Library

Bunny

Alternative Libraries

- amqp
- Hutch
- March Hare
- Sneakers

Code Sample using Bunny

Install the Library

Include the `Bunny` library in your `Gemfile` to ensure that it will be installed by bundler in CloudFoundry:

```
source 'https://rubygems.org'
```

```
ruby '>=2.3.1'
gem 'bunny', '>=2.6.4'
```

Discover the Bound Predix Message Queue Service in Ruby

Parse the `VCAP_SERVICES` environment variable to get the `predix-message-queue` service credentials using the `JSON` library. This example assumes only one `predix-message-queue`service is bound to the app.

```
vcap = JSON.parse ENV['VCAP_SERVICES']
amqp_credentials = vcap['predix-message-queue'][0]['credentials']
['protocols']['amqp']
```

Alternatively you can use the `cf-app-utils` gem:

```
credentials = CF::App::Credentials.find_by_service_label('predix-
message-queue')
amqp_credentials = credentials['protocols']['amqp']
```

Set Up the RabbitMQ Client and Strt the Connection

```
# Create the RabbitMQ connection. You can also use the URI:
#
#   conn = Bunny.new(amqp_credentials['uri'])
#
conn = Bunny.new(host: amqp_credentials['host'],
                 port: amqp_credentials['port'],
                 username: amqp_credentials['username'],
                 password: amqp_credentials['password'],
                 vhost: amqp_credentials['vhost'] )
conn.start
```

Publishing and Consuming Messages

Create the channel, exchange, and queue. Bind the queue to the exchange. In this example we use a direct exchange with a routing key. See the RabbitMQ Tutorials for more details on the different types of message routing available.

```
channel = conn.create_channel
exchange = channel.direct('sample-exchange')
queue = channel.queue('predix-message-queue-ruby-sample')
queue.bind(exchange, routing_key: 'test')
```

In your publishing application you can now use the `Queue` object to publish messages. In this case we generate a random string and publish it to the queue:

```
message = SecureRandom.uuid
STDOUT.puts "Publishing message to exchange #{exchange.name}:
#{message}"
queue.publish(message)
```

Consuming applications should discover services and set up the RabbitMQ client in the same way described above. To receive messages use the `subscribe` method of the `Queue`to define a callback message to use for handling the messages:

```
queue.subscribe(block: true, exclusive: true) do |delivery_info,
properties, message|
```

```
    STDOUT.puts "Received message from the queue: #{message}"
  end
```

Make sure to close the channel and the client connection when the app has finished using them:

```
channel.close
conn.close
```

Node.js Code Sample for using Predix Message Queue

Recommended Library

ampq.node

Alternative Libraries

rabbit.js

Code Sample using amqp.node

Install the Liberary

Install the `amplib` using npm:

```
npm init
npm install amqplib --save
```

It should also be included in your app's `package.json` file:

```
{
  "name": "nodejs-sample-consumer",
  "version": "0.1.0",
  "description": "Sample application for predix-message-queue",
  "main": "sample-consumer.js",
  "scripts": {
    "start": "node sample-consumer.js"
  },
  "author": "GE Digital",
  "license": "ISC",
  "dependencies": {
    "amqplib": "^0.5.1"
  }
}
```

Discover the Bound Predix Message Queue Service in Ruby

Parse the credentials from the app's `VCAP_SERVICES` environment variable:

```
var vcap = process.env.VCAP_SERVICES;
var vcapJSON = JSON.parse(vcap);
var credentials = vcapJSON['predix-message-queue'][0]['credentials']
['protocols']['amqp'];
```

Alternatively you can use the vcap_services library to retrieve the bound service credentials using the service and plan name:

```
var vcap_services = require('vcap_services');
var credentials = vcap_services.getCredentials('predix-message-
queue', 'Dedicated-3HA-Q20');
```

Sample App for Publishing Messages using amqplib

This example uses the `amqplib` callback API to publish a message to the Predix Message Queue service every 5 seconds using an exchange and routing key for direct routing. It will stop and close the connection after five minutes.

```
var vcap_services = require('vcap_services');
var amqp = require('amqplib/callback_api');
var uuid = require("uuid");

var credentials = vcap_services.getCredentials('predix-message-
queue', 'Dedicated-3HA-Q20');

// Error handler function logs the message and closes the
connection if there is one.
function fail(err, conn) {
    console.log(err);
    if (conn) conn.close(function() { process.exit(1); });
}

//  Generate a random UUID and publish to the exchange with the
given routing key
function publishMessage(ch, exchange, routingKey) {
    var message = uuid.v4();
    ch.publish(exchange, routingKey, Buffer.from(message));
    console.log("Publishing message to exchange %s: %s", exchange,
message);
}

// Callback function to run when the amqp connection is created.
function on_connect(err, conn) {
    if (err !== null) return fail(err);

    // Callback function to run when channel is opened
    function on_channel_open(err, ch) {
        if (err !== null) return fail(err, conn);

        // Configure the exchange, then use setInterval to publish
a message once every 5 seconds.
        var exchange = 'sample-exchange';
        ch.assertExchange(exchange, 'direct', {durable: false});
        var intervalObj = setInterval(publishMessage, 5000, ch,
exchange, 'test');

        // After five minutes clear the Interval and close down the
channel and connection
        setTimeout(function() {
            clearInterval(intervalObj);
            ch.close(function() { conn.close(); });
        }, 300000);
    }
    // Open a  channel
```

```
    conn.createChannel(on_channel_open);
}
// Connect to the predix-message-queue service using the URI
amqp.connect(credentials.uri, on_connect);
```

Sample App for Consuming Messages using amqplib

This example uses the `amqplib` callback API to consume messages from the publisher app created in . It creates a queue and binds it to the same exchange used by the publisher. It will log each message as it is received.

```
var amqp = require('amqplib/callback_api');

// Parse the VCAP_SERVICES environment variable to  JSON to get the
credentals.
var vcap = process.env.VCAP_SERVICES;
var vcapJSON = JSON.parse(vcap);
var credentials = vcapJSON['predix-message-queue'][0]['credentials']
['protocols']['amqp'];

// Define a method for error handling.
function fail(err, conn) {
    console.log(err);
    if (conn) conn.close(function() { process.exit(1); });
}


function on_connect(err, conn) {
    if (err !== null) return fail(err);
    process.once('SIGINT', function() { conn.close(); });

    conn.createChannel(function(err, ch) {
        if (err !== null) return fail(err, conn);

        var exchange = 'sample-exchange';
        var queue = 'nodejs-example-queue';

        ch.assertExchange(exchange, 'direct', {durable: false});
        ch.assertQueue(queue, {exclusive: false}, function(err, ok)
{
            if (err !== null) return fail(err, conn);
            ch.bindQueue(queue, exchange, 'test', {});

            console.log("Listening for messages on %s", exchange);
            ch.consume(queue, function(message) {
                var routingKey = message.fields.routingKey;
                var msg = message.content.toString();
                console.log("Received message with routing key %s:
%s", routingKey, msg);
            }, {noAck: true});
        });
    });
}
amqp.connect(credentials['uri']+ "?heartbeat=30", on_connect);
```

You can use `cf logs` to view log output from the app and verify that the messages are received.

Sample log output for the producer:

```
2017-08-03T13:16:08.12-0700 [DEA/6] OUT Starting app instance
(index 0) with guid 85893dae-e45f-45aa-a5b7-ea7a8dcda886
2017-08-03T13:17:30.97-0700 [API/0] OUT Updated app with guid
85893dae-e45f-45aa-a5b7-ea7a8dcda886 ({"state"=>"STARTED"})
2017-08-03T13:16:15.14-0700 [App/0] OUT Publishing message to
exchange sample-exchange: c53c3885-eba1-423b-b10d-e5ce7824de9b
2017-08-03T13:16:20.14-0700 [App/0] OUT Publishing message to
exchange sample-exchange: 06957cb9-6d94-4061-985c-2f62de9a7576
2017-08-03T13:16:25.15-0700 [App/0] OUT Publishing message to
exchange sample-exchange: 623cf6fd-ac3c-475f-813f-0ce367b4025a
2017-08-03T13:16:30.15-0700 [App/0] OUT Publishing message to
exchange sample-exchange: ced5af35-9012-4ed0-8b27-45e1fb6a9691
2017-08-03T13:16:35.16-0700 [App/0] OUT Publishing message to
exchange sample-exchange: 8d015070-3095-430f-bb1e-9cdf4d672f34
```

Sample log output fro the consumer:

```
2017-08-03T13:15:47.90-0700 [DEA/1] OUT Starting app instance
(index 0) with guid f9fb69f2-5e83-4c70-a44e-10659cd23c71
2017-08-03T13:15:49.98-0700 [App/0] OUT Listening for messages on
sample-exchange
2017-08-03T13:16:40.05-0700 [App/0] OUT Received message with
routing key test: c53c3885-eba1-423b-b10d-e5ce7824de9b
2017-08-03T13:16:45.06-0700 [App/0] OUT Received message with
routing key test: 06957cb9-6d94-4061-985c-2f62de9a7576
2017-08-03T13:16:50.06-0700 [App/0] OUT Received message with
routing key test: 623cf6fd-ac3c-475f-813f-0ce367b4025a
2017-08-03T13:16:55.07-0700 [App/0] OUT Received message with
routing key test: ced5af35-9012-4ed0-8b27-45e1fb6a9691
2017-08-03T13:17:00.07-0700 [App/0] OUT Received message with
routing key test: 8d015070-3095-430f-bb1e-9cdf4d672f34
```

Python Code Sample for Predix Message Queue

Recommended Library

pika

Alternative Libraries

- Celery
- Haigha

Code Sample using pika

Install the Library

To ensure that CloudFoundry will install the pika module add a `requirements.txt` to your app directory that contains the following line:

```
pika>=0.10.0
```

Discover the Bound Predix Message Queue Service in Python

You can use os.environ and the `json` library to parse the `VCAP_SERVICES` environment variable. If the variable isn't defined, or if the service isn't found, a `KeyError` will be thrown.

```
import sys
import json
from os import environ

try:
    vcap_services = json.loads(environ['VCAP_SERVICES'])
    amqp_credentials = vcap_services['predix-message-queue'][0]
['credentials']['protocols']['amqp']
except KeyError as error:
    print("Could not parse credentials from the environment. Key
not found: {0}".format(error))
    sys.exit(1)
```

You can also use the cfenv library to parse the app's credentials:

```
from cfenv import AppEnv

app_env = AppEnv()
rabbit_service = app_env.get_service(label='predix-message-queue')
amqp_credentials = rabbit_service.credentials['protocols']['amqp']
```

Set Up the Ra bbitMQ Connection

Use the `PlainCredentials` and `ConnectionParameters` classes to set up the connection configuration using the service credentials, then use them to open the connection and create a channel.

```
# Configure the authentication credentials
rabbit_creds = pika.PlainCredentials(amqp_credentials['username'],
amqp_credentials['password'])

# Set up connection parameters
params = pika.ConnectionParameters(host=amqp_credentials['host'],
                                   port=amqp_credentials['port'],

virtual_host=amqp_credentials['vhost'],
                                   ssl=amqp_credentials['ssl'],
                                   credentials=rabbit_creds)

# Open the connection and create a channel
connection = pika.BlockingConnection(params)
channel = connection.channel()
```

Publishing Messages

Use the `basic_publish` method of the `Channel` to publish a message. This example uses direct routing with a routing key:

```
channel.basic_publish(exchange='sample-exchange',
routing_key='test', body='Hello World!')
```

Consuming Messages

To consume messages first declare a queue and an exchange, and then bind them with a routing key:

```
To consume messages first declare a queue and an exchange, and then
bind them with a routing key:
```

Then define a callback method to handle incoming messages, and pass it to the `basic_consume` method of the `Channel`:

```
# Define a callback method to use for handling messages
def callback(ch, method, properties, body):
    print("Received message with routing key {0}:
{1}".format(method.routing_key, body))

# Set up the consumer and begin listening for messages.
channel.basic_consume(callback, queue=queue_name, no_ack=True)
channel.start_consuming()
```

Full Sample Producer App

Here's the full example publisher which will publish a random UUID to the exchange once every five seconds for five minutes:

requirements.txt

```
pika>=0.10.0
```

manifest.yml

```
applications:
- name: python-sample-producer
  command: python sample-producer.py
  no-route: true
  mem: 256M
  disk: 1024M
  instances: 1
```

Procfile

```
web: python sample-producer.py
```

sample-producer.py

```
import sys
import json
import pika
from os import environ
from uuid import uuid4
from time import sleep

# Parse the credentials from the VCAP_SERVICES environment
variable
try:
    vcap_services = json.loads(environ['VCAP_SERVICES'])
    amqp_credentials = vcap_services['predix-message-queue'][0]
['credentials']['protocols']['amqp']
```

```
    # Configure the authentication credentials
    rabbit_creds =
pika.PlainCredentials(amqp_credentials['username'],
amqp_credentials['password'])

    # Set up connection parameters
    params =
pika.ConnectionParameters(host=amqp_credentials['host'],

port=amqp_credentials['port'],

virtual_host=amqp_credentials['vhost'],

ssl=amqp_credentials['ssl'],
                                        credentials=rabbit_creds)

except KeyError as error:
    print("Could not parse credentials from the environment. Key
not found: {0}".format(error))
    sys.exit(1)

# Open the connection and create a channel
connection = pika.BlockingConnection(params)
channel = connection.channel()

# Publish a message to the queue every 5 seconds for 5 minutes.
max_seconds = 300
total_seconds = 0
while total_seconds < max_seconds:
    # Generate a random UUID string to use as the message.
    message = str(uuid4())
    # Publish the message to the channel using basic_publish with
an exchange and routing key
    channel.basic_publish(exchange='sample-exchange',
routing_key='test', body=message)
    print("Published message to exchange sample-exchange:
{0}".format(message))
    sleep(1)
    total_seconds += 1

connection.close()
sys.exit(0)
```

Sample log Output:

```
2017-08-04T10:48:09.28-0700 [DEA/5] OUT Starting app instance
(index 0) with guid 0b803338-8c3a-4620-b6c7-73bdc2de9efa
2017-08-04T10:48:11.83-0700 [App/0] OUT Published message to
exchange sample-exchange: 5769f6a3-2587-4202-b5c4-050aaf8ea16f
2017-08-04T10:48:12.83-0700 [App/0] OUT Published message to
exchange sample-exchange: e83743d8-9d04-4c44-8f88-b3216b890b8d
2017-08-04T10:48:13.83-0700 [App/0] OUT Published message to
exchange sample-exchange: 655b684a-8aab-41c3-8370-5dfe9ea8b64a
2017-08-04T10:48:14.83-0700 [App/0] OUT Published message to
exchange sample-exchange: 19bbd3b9-dfc9-4c68-b854-ef68f2f82440
```

Full Sample Consumer App

This sample consumer app will retrieve the messages published by the above producer app and print them to the app's logs. It also demonstrates using the cfenv library to parse the service details from the environment.

requirements.txt

```
pika>=0.10.0
```

manifest.yml

```
applications:
- name: python-sample-consumer
  command: python sample-consumer.py
  no-route: true
  mem: 256M
  disk: 1024M
  instances: 1
```

Procfile

```
web: python sample-consumer.py
```

sample-producer.py

```
import sys
import json
import pika
from cfenv import AppEnv

app_env = AppEnv()
rabbit_service = app_env.get_service(label='predix-rabbitmq')
amqp_credentials = rabbit_service.credentials['protocols']['amqp']

# Configure the authentication credentials
rabbit_creds =
pika.PlainCredentials(amqp_credentials['username'],
amqp_credentials['password'])

# Set up connection parameters
params = pika.ConnectionParameters(host=amqp_credentials['host'],
                                   port=amqp_credentials['port'],

virtual_host=amqp_credentials['vhost'],
                                   ssl=amqp_credentials['ssl'],
                                   credentials=rabbit_creds,
                                   heartbeat_interval=30)


# Open the connection and create a channel
connection = pika.BlockingConnection(params)
channel = connection.channel()

# Declare the exchange and the queue. Bind the queue to the
exchange with a routing key.
queue_name = 'predix-message-queue-python-sample'
channel.exchange_declare(exchange='sample-exchange',
type='direct')
```

```
channel.queue_declare(queue=queue_name, exclusive=False)
channel.queue_bind(exchange='sample-exchange', queue=queue_name,
routing_key='test')


# Define a callback method to use for handling messages
def callback(ch, method, properties, body):
    print("Received message with routing key {0}:
{1}".format(method.routing_key, body))

print("Listening for messages on queue {0}".format(queue_name))
# Set up the consumer and begin listening for messages.
channel.basic_consume(callback, queue=queue_name, no_ack=True)
channel.start_consuming()
```

Sample Log Output

```
2017-08-04T10:47:10.97-0700 [DEA/10] OUT Starting app instance
(index 0) with guid f37adf39-15f0-45cb-9028-b9ebe9981cd3
2017-08-04T10:47:13.46-0700 [App/0] OUT Listening for messages on
queue predix-messagae-queue-python-sample
2017-08-04T10:48:11.58-0700 [App/0] OUT Received message with
routing key test: 5769f6a3-2587-4202-b5c4-050aaf8ea16f
2017-08-04T10:48:12.59-0700 [App/0] OUT Received message with
routing key test: e83743d8-9d04-4c44-8f88-b3216b890b8d
2017-08-04T10:48:13.59-0700 [App/0] OUT Received message with
routing key test: 655b684a-8aab-41c3-8370-5dfe9ea8b64a
2017-08-04T10:48:14.59-0700 [App/0] OUT Received message with
routing key test: 19bbd3b9-dfc9-4c68-b854-ef68f2f82440
```

PHP Code Sample for Predix Message Queue

Recommended Library

php-amqplib

Alternative Libraries

- VorpalBunny
- PECL AMQP Library
- Thumper

Code Sample using amqplib

Install the Library

To ensure that CloudFoundry will install `php-amqplib` when the app is uploaded add the following entry to the `composer.json` file:

```
{
  "require": {
    "php-amqplib/php-amqplib": ">=2.6.1"
  }
}
```

Make sure to include the `autoload.php` file from the `vendor` directory in order to include use the library:

```
<?php
require_once realpath(__DIR__ . DIRECTORY_SEPARATOR . '../lib/
vendor'). '/autoload.php';
?>
```

Discover the Bound Predix RabbitMQ Service in PHP

GO Code Sample for Predix Message Queue

Recommended Library

Go RabbitMQ

Alternative Libraries

Rabbit Hole (RabbitMQ HTTP API client for GO)

Code Sample using GO RabbitMQ

Install the Library

Install the library using `go get`:

```
go get github.com/streadway/amqp
```

If you're using `Godep` save the dependencies:

```
godep save
```

For `Glide` make sure to add the library to your `glide.yaml` file:

```
package: github.com/ge-digital/rabbitmq-sample-go
import:
- package: github.com/streadway/amqp
```

Make sure to import the library into your application:

```
package main

import (
    "fmt"
    "os"

    "github.com/streadway/amqp"
)
```

Discover the Bound Predix Message Queue Service in GO

You can use `os.Getenv` and the `encoding/json` library to parse the credentials from the `VCAP_SERVICES` environment variable:

```
package main
```

```
import (
    "os"
    "encoding/json"
)

func failOnError(err error, message string) {
  if err != nil {
    fmt.Println(fmt.Sprintf("%s: %s\n", message, err))
    os.Exit(1)
  }
}

func main() {
    // Parse the VCAP_SERVICES variable to map[string]interface{}
with json
  vcapServicesString := os.Getenv("VCAP_SERVICES")
  var services interface{}
  err := json.Unmarshal([]byte(vcapServicesString), &services)
  failOnError(err, "Failed to parse VCAP_SERVICES to JSON")

  // Get the amqp protocol credentials
  amqpCreds, ok := services.
           (map[string]interface{})["predix-rabbitmq"].
           ([]interface{})[0].
           (map[string]interface{})["credentials"].
           (map[string]interface{})["protocols"].
           (map[string]interface{})["amqp"].
                   (map[string]interface{})

  if !ok {
    fmt.Println("Failed to parse credentials from the VCAP_SERVICES
environment.")
    os.Exit(1)
  }
}
```

Alternatively, you can use the go-cfenv library:

```
func main() {
    // Get the app's environment and then retrieve the bound
services info for predix-message-queue services
    appEnv, err := cfenv.Current()
    failOnError(err, "Failed to read CloudFoundry environment")

    // Parse the amqp credentials from the bound services. Assumes
only one instance is bound.
    services, err := appEnv.Services.WithLabel("predix-message-
queue")
    failOnError(err, "Failed to find service with label predix-
message-queue")
    amqpCreds := services[0].Credentials["protocols"].
(map[string]interface{})["amqp"].(map[string]interface{})
}
```

Set Up the RabbitMQ Connection

Create the connection and channel, and declare an exchange:

```
// Open a connection to RabbitMQ using the URI
conn, err := amqp.Dial(amqpCreds["uri"].(string))
failOnError(err, fmt.Sprintf("Failed to connect to RabbitMQ with
uri %v", amqpCreds["uri"]))
defer conn.Close()

// Declare a channel
channel, err := conn.Channel()
failOnError(err, "Failed to open a channel")
defer channel.Close()

// Declare a direct exchange
err = channel.ExchangeDeclare(
    "sample-exchange", // name
    "direct",          // type
    false,             // durable
    false,             // autoDelete
    false,             // internal
    false,             // noWait
    nil,               // args
)
failOnError(err, "Failed to declare an exchange")
```

Publishing Messages

```
// Convert the message string to []byte
message := "Hello!"
body := []byte(message)

// Publish a message
err := channel.Publish(
    "sample-exchange", // exchange
    "test",                 // routing key
    false,             // mandatory
    false,             // immediate
    amqp.Publishing{
        ContentType: "text/plain",
        Body: body,
    })
failOnError(err, "Failed to publish message")
```

Consuming Messages

To consume messages first declare a queue, then bind it to the exchange. This example creates a goroutine to handle incoming messages, and uses an unbuffered channel to block the thread while waiting to receive them.

```
queue, err := channel.QueueDeclare(
    "predix-rabbitmq-go-sample", // name
    false,                       // durable
    false,                       // autoDelete
    true,                        // exclusive
    false,                       // noWait
    nil,                         // args
)
```

```
failOnError(err, "Failed to declare a queue")

// Bind the queue to the exchange with a routing key
err = channel.QueueBind(queue.Name, "test", "sample-exchange",
false, nil)
failOnError(err, "Failed to bind the queue")

messages, err := channel.Consume(
    queue.Name,  // queue name
    "",          // consumer
    true,        // autoAck
    false,       // exclusive
    false,       // noLocal
    false,       // noWait
    nil,         // args
)
failOnError(err, "Failed to register the consumer")

// Declare an empty channel to use to block the thread while
waiting for messages
forever := make(chan bool)

// Use a goroutine to handle incoming messages
go func() {
    for m := range messages {
        out := fmt.Sprintf("Received message from exchange %v with
routing key '%v': %s",
                           m.Exchange, m.RoutingKey,
string(m.Body))
        fmt.Println(out)
    }
}()

fmt.Println(fmt.Sprintf("Listening for messages on %v", queue.Name))
<-forever // Listen on the channel to block the thread
```

Backups for Predix Message Queue

Backup Overview

The Predix Message Queue service instance configuration is exported and saved on an hourly basis.

What is Backed Up

Current RabbitMQ Configuration Data

The backup process exports the current server state using data returned by calls made to the `/api/definitions/<vhost>` Management API REST interface. Data retuned from his API call will include the following information:

Server definitions including

- Virtual host definitions
- Exchange definitions
- Queues
- Bindings

- Policies

  Policies will be exported using data returned from calls made to the `/api/policies`endpoint of the Management API.

What is Not Backed Up

RabbitMQ doesn't provide an interface to backup or export data stored in 'Durable Queues'.

Failure Scenarios in which Data Loss is Possible (Durable Queues)

Single Node plans, when a VM's data partition is destroyed, RabbitMQ server configuration will be restored, however, the data partition will be empty and can't be restored. Durable Queue Data will be lost!

HA Plans, when 2 or more nodes fail and data partitions are destroyed, RabbitMQ server configuration will be restored, however, the data partition will be empty and can't be restored. Durable Queue Data will be lost!

Recovery Time Objective (RTO)_

The Recovery Time Objective for the Predix Message Queue is four hours. In the event of a catastrophic failure, your instance should be restored to service within 4 hours.

Recovery Point Objective (RPO)

The Recovery Point Objective for the Predix Message Queue is one hour. In the event of a catastrophic failure in which data recovery is necessary, you should expect no more than 60 minutes of RabbitMQ Server configuration data loss. See failure scenarios which detail events which could contribute to data loss.

Migrating from an Existing Service or Plan to a New Plan

Migrate your messaging to a new service plan with Predix Message Queue.

Migrating Data to the New Service

Currently there is no supported method for migrating any existing state or messages from another RabbitMQ service. You will need to recreate any Exchanges, Queues, Bindings, or Policies you wish to use with the new service.

Changing from an Existing Plan to a New Plan

To migrate from a previous RabbitMQ plan perform the steps provided here. A brief downtime for your application will be required, so plan the migration accordingly.

Procedure

1. Create a new Predix Message Queue service instance. See Creating a Predix Message Queue Service Instance on page 2 for details.
2. Stop the app.
3. Unbind the existing RabbitMQ service instance that you want to replace with Predix Message Queue.
4. Bind the new Predix Message Queue service instance to the app.
5. Restart the application.