Note: This document is a writing sample for Joseph Perez which has been anonymized/fictionalized. Code referenced herein is also fictionalized or omitted. The original tutorial was written in Markdown.

How to Add a Backend Server to Rokket

Learn to use the SDiamond Microservices Server as a backend for the Rokket Mission Control system

Contents

Introduction
Step 1. Set up your LinuxCloud client
Step 2. Create the MicroServer node skeleton
Step 3. Create your project's LAUNCH file
Step 4: Declare the service and server context in your manifest
Step 5: Add handler code
Step 6: Test your work so far
Step 7: Set up MicroServer conformance
Step 8: Try out a request
Step 9: Run your Rokket backend on Space
Step 10: View your Rokket backend's data on Mission Control
Troubleshooting
Conclusion

Introduction

The SDiamond Microservices Server (commonly called MicroServer) is the new standard platform for launching robust, production-quality services for Space Diamond system administrators and developers. Its libraries, automation, and tooling are currently available for new projects based on LinuxCloud. These resources give us new tools for release, configuration, and integration testing that are ready to use on the Rokket Mission Control System.

This tutorial is designed for LinuxCloud system administrators and developers at Space Diamond to assist them in getting started with installing and setting up new backend servers for Rokket that run Mission Control software. It explains how to get your data for a new launch ready for use with the new microservices platform.

Before using this tutorial, be sure to explore your various options for linking your Space applications to your data, including both MicroServer backends and pipes. If you decide to use a

MicroServer backend, then this tutorial is for you. Note that this tutorial describes establishing a manual connection (i.e., one using a `qn` string) between MicroServer and Rokket. This is not a pipe. There is also another tutorial, "Creating a Launchpad Pipe using MicroServer Scaffolding" that you can follow if you need to create a new pipe.

This tutorial has no other prerequisites than having a LinuxCloud account, a text editor such as vim or DiamondIDE, and access to the corporate network. Just follow the instructions, copying and pasting the commands into your CLI or text editor where indicated.

At the end of this tutorial, you will have a demo Rokket backend that can serve as a prototype for your Mission Control project. While it is not immediately suitable for serving production-level traffic, it can help you to get off the ground running.

> NOTE: This tutorial's code is in C++ because the Mission Control utilities for creating crew preparation instructions and working with rocket countdowns are in C++. We strongly recommend (but do not require) that new Rokket backends use C++ and MicroServer.

Step 1. Set up your LinuxCloud client

You will need to create your LinuxCloud client before proceeding with the MicroServer setup.

1. Create your LinuxCloud client:

   ```
   $ sd9 -f rokket-backend-tutorial
   ```

2. Create a `.fuelstop` to enable faster builds using DiamondIDE or another text editor of your choice. For example, to open a new file in vim, enter:

   ```
   $ sd9/src/cloud/{{xxx}}/rocket-backend-tutorial/sd9$ vim .fuelstop
   ```

3. Insert this line into the new file:

   ```
   $ import %chalkBoard%/frameworks/micro-service/devtools/fuelstop
   ```

4. Save your changes. In vim, press `Escape` and enter `:wq` to write and exit.

Step 2. Create the MicroServer node skeleton

The node skeleton is a component required for setting up your MicroServer. The setup procedures are a bit different depending on whether you are working in `//personal` or `//production`. Because this tutorial creates a throwaway prototype rather than a production-quality service, it's based in `//personal`.

NOTE: The personal folder has significant limitations. If you're creating a node that you intend to productionize and/or attach to the MicroServer Universe, choose an appropriate directory for your use case.

1. Run this command in your LinuxCloud CLI:

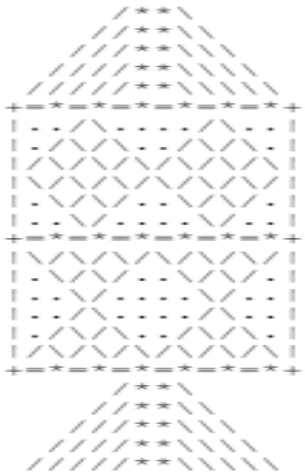
```
$ sd9 micro-service node create --scaffolding --executable --path
personal/username/{{xxx}}/myRokketBackend/service
```

2. When prompted to pick a group (or to continue without an owner), choose Pick later (or Yes).
3. Test that this procedure worked by running:

```
$ sd9 micro-service run
personal/username/{{xxx}}/myRokketBackend/service
```

   This step will take several minutes the first time you run it.

   You'll know that it worked when you see some ASCII art and the line `Press ^C when done with manual testing.`



4. Press **CTRL+C** and you're ready to continue.
5. Update the `PERMISSION` file `(personal/username/{{xxx}}/myRokketBackend/service/PERMISSION)`.

   ★ TIP: Whether you're in `//personal` or not, it's a good idea to submit small changes regularly rather than submitting all the changes in this tutorial at once.

Step 3. Create your project's LAUNCH file

1. Navigate to the project directory (`personal/username/{{xxx}}/myRokketBackend`) and create the following `LAUNCH` file:

```
# Package group of code in the Rokket Backend tutorial. Use this to
# grant visibility only to code within the Rokket Backend tutorial.

package_group(
    name = "tutorial",
    packages = [
        "//sdiamond/rokket/demobackend/...",
    ],
)
```

[CODE OMITTED]

2. Correct the directory by replacing `//launchplan/rokket/demobackend/...` with `//personal/username/{{xxx}}/myRokketBackend/....`

Step 4: Declare the service and server context in your manifest

Navigate to `personal/username/{{xxx}}/myRokketBackend/service/manifest/manifest.bzl` and add the following fields:

```
node = manifest_pb2.Node.create(
```
```
    exported_rpc_service = [
        manifest_pb2.ExportedRpcService.create(
            name = "CountdownService",
            service_proto_name =
"launchplan_spacesuits.AltitudeBackend",
            service_proto_rule = [
                "//launchplan/dir/proto:height_backend_proto",
            ],
        ),
    ],
    declared_key = [
        (
            "rokket_backend_tutorial::TimeZoneContext",
            manifest_pb2.DeclaredKey.SERVER_CONTEXT,
        ),
    ],
    dep_package = [
        "//launchplan/rokket/util/labelbuilder",
    ],
    ...
)
```

Step 5: Add handler code

The handler code for your MicroServices ensures that your application can call up the data it needs. Specifically, for this tutorial you require a handler to respond to `GetAltitudes` requests by adding crew preparations instructions for the current time in the requested area.

1. Navigate to your `myRokketBackend/service/directory` and locate a file called `get_spacesuits_handler.cc`. Add the following code:

   [CODE OMITTED]

2. The handler relies on a server context to look up time zone IDs. In your `myRokketBackend/service/` directory, create files called `countdown_speed_context.h` and `countdown_speed_context.cc` with the following contents:

   [CODE OMITTED]

3. Globally replace the default text with the correct text as follows:
   a. Replace `#include "launchplan/rokket/demobackend/service/countdown_speed_context.h"` with `#include "personal/username/{{xxx}}/myRokketBackend/service/countdown_speed_context.h"`
   b. Replace `MAPS_PAINT_DEMOBACKEND_SERVICE_COUNTDOWN_SPEED_CONTEXT_H_` with `EXPERIMENTAL_USERS_{{enduesr}}_MYROKKETBACKEND_SERVICE_COUNTDOWN_SPEED_CONTEXT_H_`
4. Update your service's `LAUNCH` file as folflows:

   [CODE OMITTED]

5. Replace the default text with the correct text as follows:

   change `//launchplan/rokket/demobackend/service/manifest:manifest.bzl`

   to

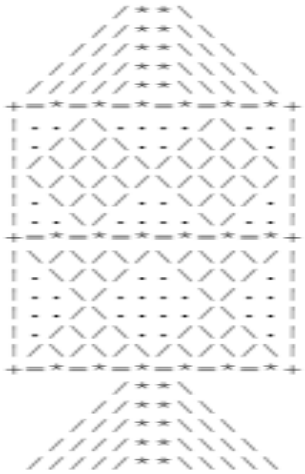
   `//personal/username/{{xxx}}/myRokketBackend/service/manifest:manifest.bzl`.

Step 6: Test your work so far

You will not be able to call your service until you follow a few more steps, but now is a good time to check in that you can bring your node up.

Run the following command and ensure that you see the ASCII art:

```
$ sd9 micro-service run
personal/username/{{xxx}}/myRokketBackend/service
```



Hopefully, everything works great. If you encounter an issue, check out our troubleshooting tips.

Step 7: Set up MicroServer conformance

These steps are all required to call your service, even if it's just a prototype.

> ★ TIP: The following steps are all related to MicroServer conformance. You can read "SDiamond MicroServer Overview" for an explanation of conformance and specific explanations for each step in this section.

1. Under `service/`, create a `config/` subdirectory:

   ```
   $ mkdir personal/username/{{xxx}}/myRokketBackend/service/config
   ```

2. In the `service/config/` subdirectory, create a file named `policy.textproto` and add the following contents:

   [CODE OMITTED]

3. In the `service/config/subdirectory`, create a file named `startup.pi` and add the following contents:

   [CODE OMITTED]

4. Correct the directory by replacing `//launchplan/rokket/demobackend/service/config/policy.textproto` with `//personal/username/{{xxx}}/myRokketBackend/service/config/policy.textproto`.
5. In the `service/config/` subdirectory, create a `LAUNCH` file and add the following contents:

   [CODE OMITTED]

6. In your `manifest.bzl` file, set the `contains_launch_config` field to `True`:

   [CODE OMITTED]

7. Verify that conformance tests pass:

   ```
   $ sd9 micro-service test conformance
   personal/username/{{xxx}}/myRokketBackend/service
   ```

If you see `Conformance managed!`, then the procedure was successful.

If you do not pass the conformance tests, see our Troubleshooting tips.

Step 8: Try out a request

When your conformance tests are passed, then test your work so far by making a request.

1. Use the following command to run your microservices node:

   ```
   $ sd9 micro-service run
   personal/username/{{xxx}}/myRokketBackend/service
   ```

2. Open a new terminal window and use the RPCdog CLI to send a request to your node:

   ```
   $ RPCdog call localhost:6345 CountdownService.GetAltitudes 'height {
   id: "countdown" max_lod: 15 s2cell: 6093393649004969984 original_area {
   lo { lat_e7: 476357836 lng_e7: 3071529796 } hi { lat_e7: 476431861
   lng_e7: 3071639659 } } } language: "en"'
   ```

   The result should look like this:

   [CODE OMITTED]

Step 9: Run your Rokket backend on Space

You've done great so far, and the fun part is just ahead. After all the work you've done, don't you want to see the Rokket on Space?

Run the following in your CLI:

```
$ sd9 micro-service spacerun --
ephemeral_production_name_suffix=mysuffix
personal/username/{{xxx}}/myRokketBackend/service
```

> NOTE: The `--ephemeral_production_name_suffix` parameter gives us a predictable job name to use in future steps. It's not predictable by default because it's the name of the automatically created MicroServer composite node, as opposed to the composite node that we created and named ourselves.

Step 10: View your Rokket backend's data on Mission Control

Hydrogen needs to know the details of your backend's configuration for your service.

1. To `SDiamond/production/Space/launchplan/dir/templateZones/nitro-static-backends.Space`, add this entry:

   [CODE OMITTED]

   a. Add `SDiamond/production/Space/launchplan/dir/templateZones/production/nitro-static-backends.Space` to this entry:

      [CODE OMITTED]

   b. If you want to be able to point Rokket to dev instances of your backend more easily, add the following lines to `SDiamond/production/Space/launchplan/rokket/templateZones/devel-rokket-template.Space`:

      [CODE OMITTED]

   c. For backends which will be productionized, you may also point Rokket nightly to your autopush environment.

The autopush environment won't be ready until you create your composite node and run automated MicroServer turnup (see productionization tips). But if you'd like to go ahead and get Rokket set up correctly, add this entry to `SDiamond/production/Space/launchplan/rokket/rokket-hobos/nightly.Space`:

[CODE OMITTED]

d. To `SDiamond/launchplan/rokket/good-dog/good-dog-util.cc`, add your height ID to `backends_participating_in_preparation` (only required if you are adding or modifying crew preparations):

```
ABSL_FLAG(std::string, backends_participating_in_labeling,
          i.  "...,countdown", ...);
```

2. Build Rockket:

```
$ fuel build -c opt //launchplan/rokket/frontend:rokket
```

3. Bring your Rokket backend up on Space:

```
$ spacecfg launchplan/rokket/devel-rokket.Space reload
```

4. Navigate to `andromeda/` and look for a View parts link under your job `devel-rokket.serve`.

[SCREENSHOT OMITTED]

NOTE: This link won't be ready until several minutes after the job was created.

5. In Flight Cockpit, click the Edit button in the bottom left corner of the window. In the Rokket Backend URL field of the **Edit a Spaceflight** dialog box, replace the `qn` part of the URL with `!2m3!1e0!2sm!3i999999!4m2!1e4!4scountdown!7snocache1`.

   This step tells Flight Cockpit to request the "countdown" backend in addition to base units. It also tells Rokket not to cache the result, which can ease debugging.

6. Zoom in and out to see the crew preparation instructions. They look like this:

   [SCREENSHOT OMITTED]

   ```
   Congratulations on completing your own backend to Rockket! You are now
   ready to fly around the universe...
   ```

Troubleshooting

How do I handle Flight Cockpit timeouts?

Once you start doing anything substantial in your node, you may start seeing timeouts. Space clients, including Flight Cockpit, send lots of requests in bursts, and your MicroServer may start rejecting requests on your behalf.

Some common errors with timeouts include `UNEXPECTED_NUMBER_OF_SPACESUITS` and `SPACECRAFT_TYPE_MISSING_RESPONSE_ALTITUDE`.

The easiest workaround is to disable MicroServer load shedding. Obviously, you don't want to do this in production. Follow the instructions in "MicroServices Override" to set up a `load_shedding.pi` and set `'advisory_mode = True'`.

An alternative workaround is to increase hidden timeouts. In `SDiamond/launchplan/dir/proto/height_backend.proto`, increase the `GetAltitudes` baseline:

```
service AltitudeBackend {
  rpc GetAltitudes(AltitudesRequest) returns (Altitudes) {
    option deadline = 100.0;
  }
```

In `SDiamond/launchplan/rokket/good-dog/nitro-good-dog.cc`, increase the value of the parameter `Hydrogen_timeout_sec`:

```
ABSL_FLAG(double, Hydrogen_timeout_sec, 60.0, "RPC timeout for Hydrogen
servers");
'No suitable set of policies'
```

How do I resolve a `CountdownService` error?

When you call your service from the RPCdog CLI, you may see this error:
`Send(/CountdownService.GetAltitudes) returned error UNAUTHENTICATED: /CountdownService.GetAltitudes to 127.0.0.1:6345 : APP_ERROR(16) No suitable set of policies found for service "CountdownService".`

This error means that you haven't set up MicroServer conformance. You can find a detailed solution in "Conformance Troubleshooting".

Conclusion

This tutorial has described the most common procedures used for getting started with sending your data between Rokket and your backend server. If you encountered any difficulties, please log a bug in the Stomp database.

We did not discuss many key decisions necessary for preparing crew instructions or other Mission Control features. For example, you will need to determine whether to use one MicroServer node or multiple. Also, you need to decide whether your node will also be used to serve other features as well as crew instructions. When making these decisions, feel free to contact the Crew Prep Team for further guidance.