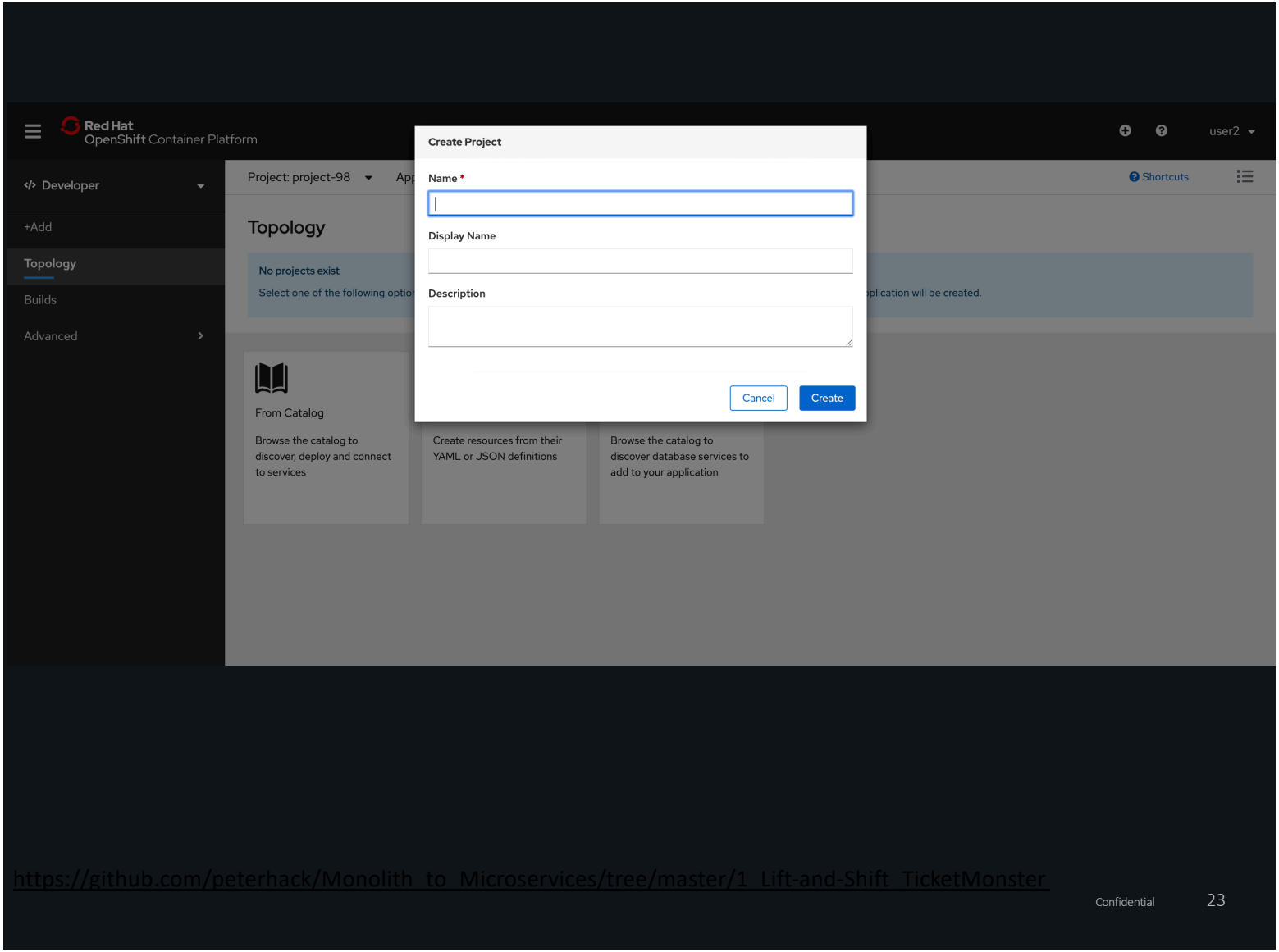
## Step 1:

## Setup Project

a. **Login to OpenShift**

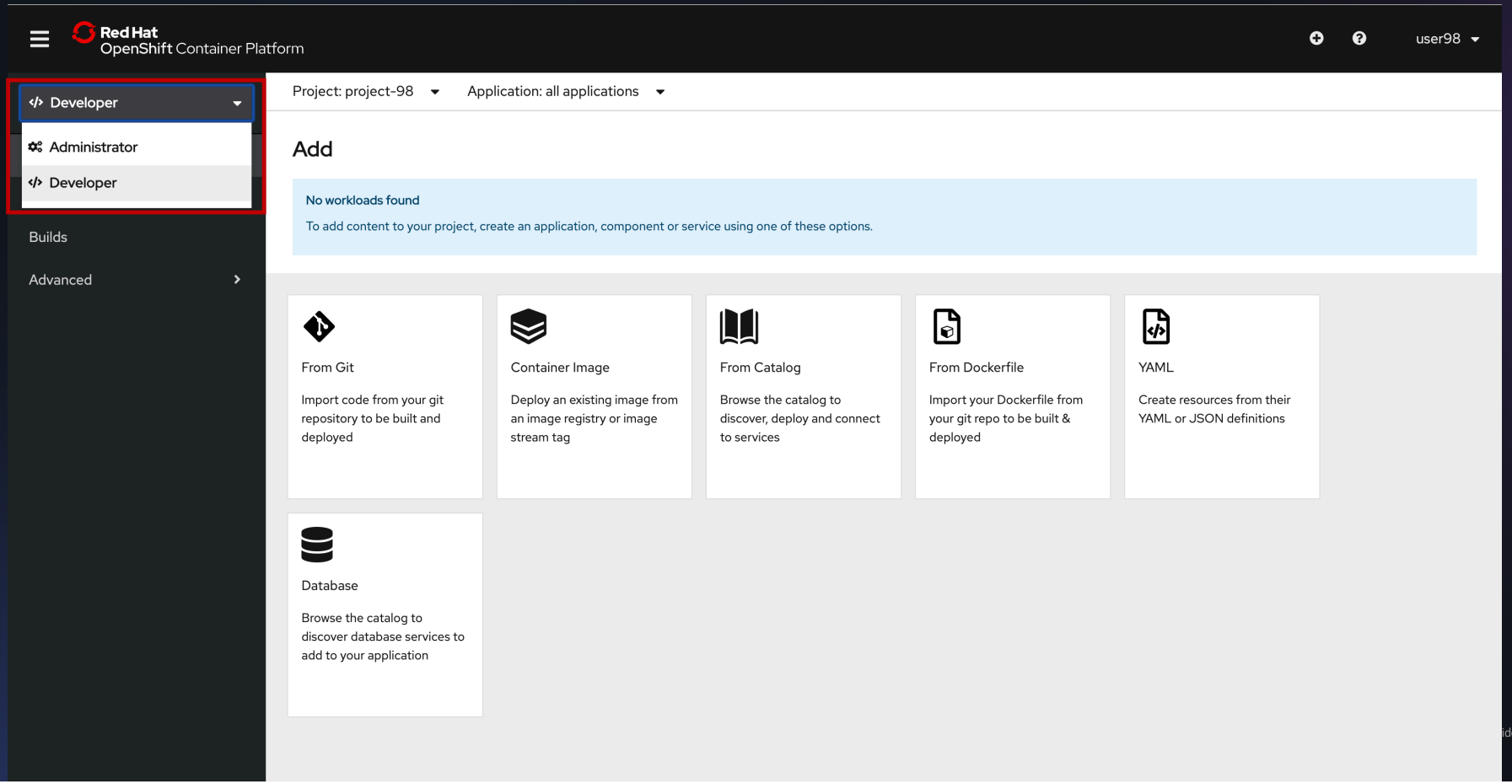
b. **Create Project-##**

c. **Navigate to Project**

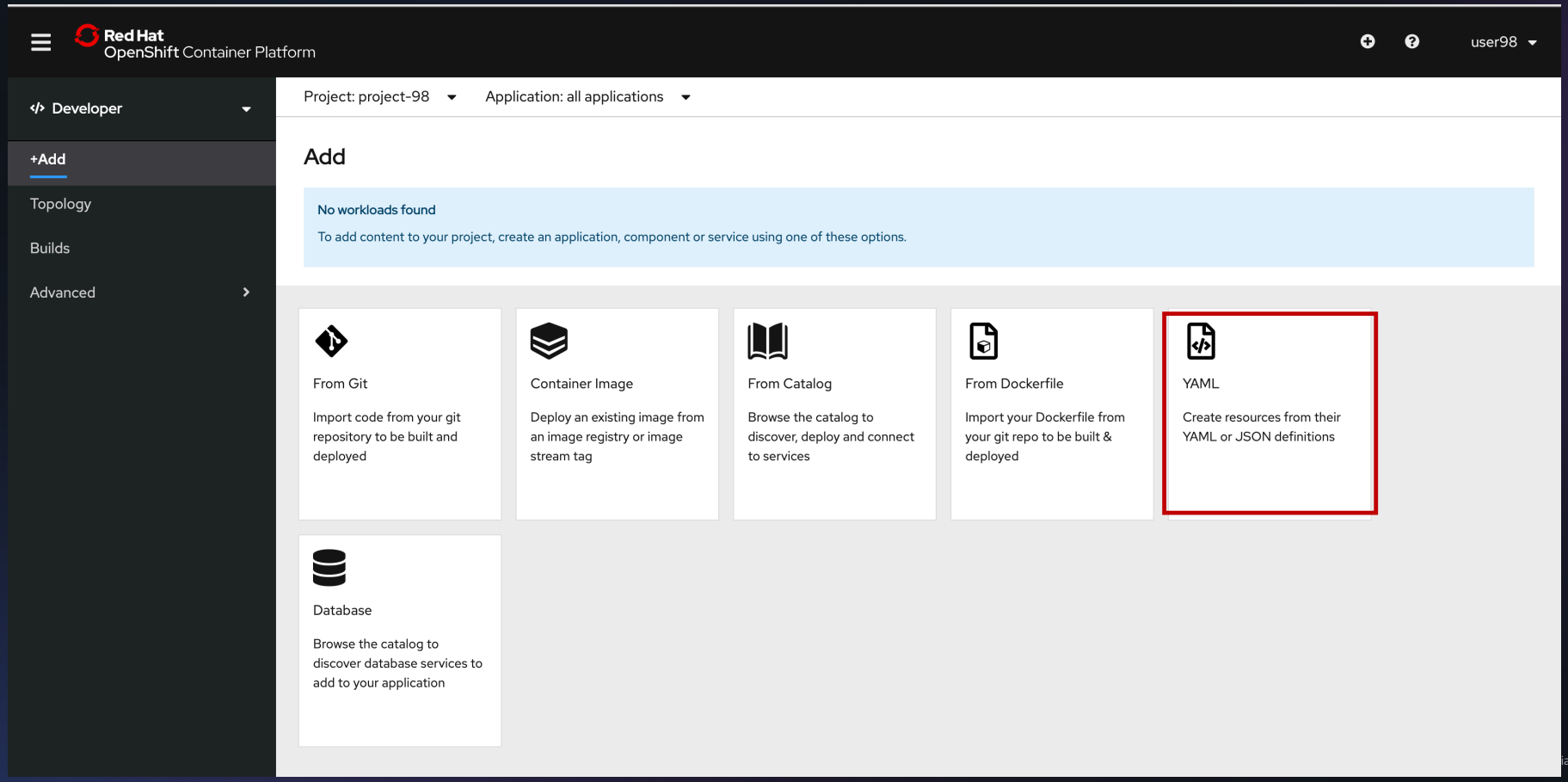



https://github.com/peterhack/Monolith_to_Microservices/tree/master/1_Lift-and-Shift_TicketMonster

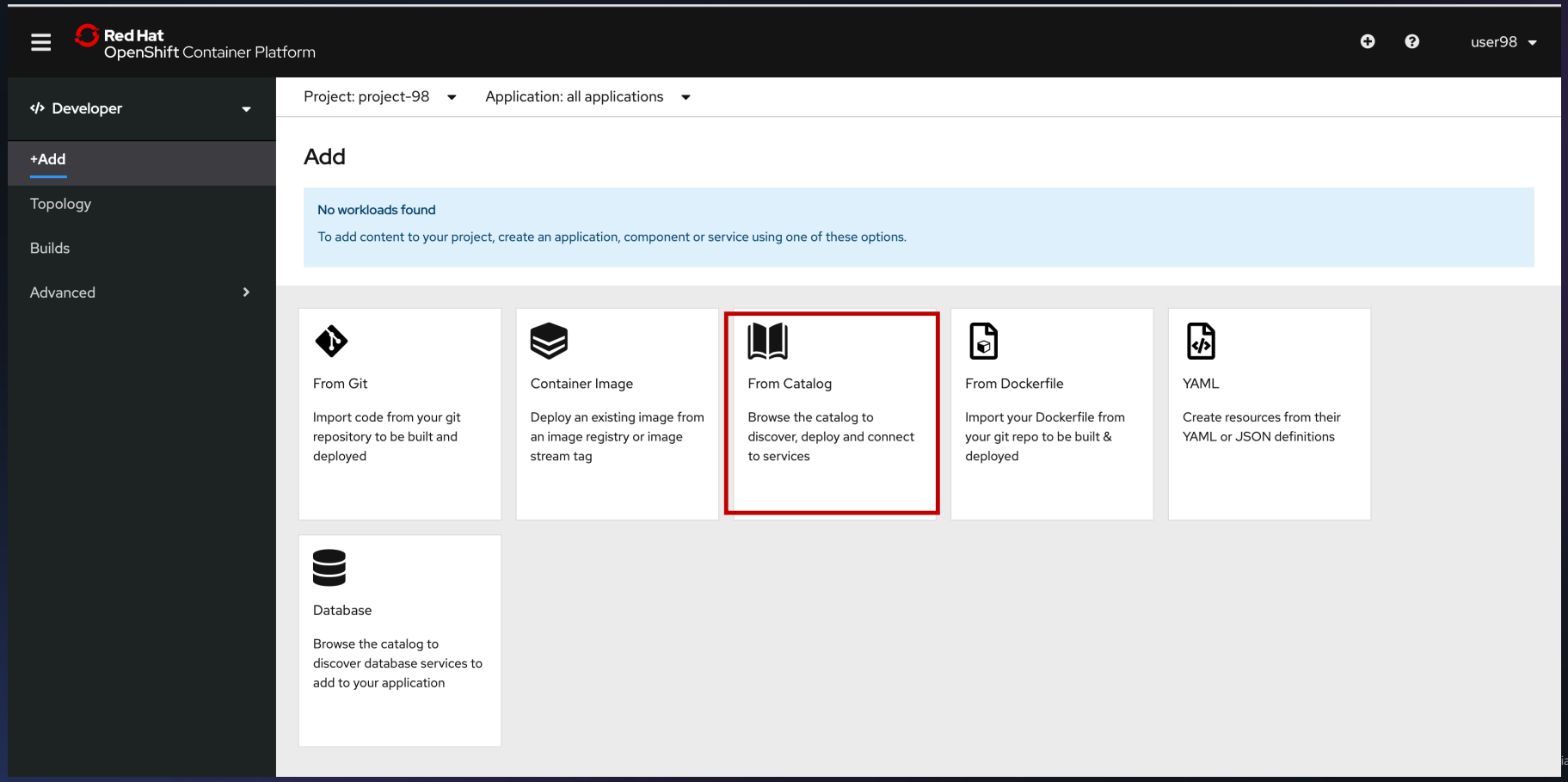
# Openshift UI Navigation



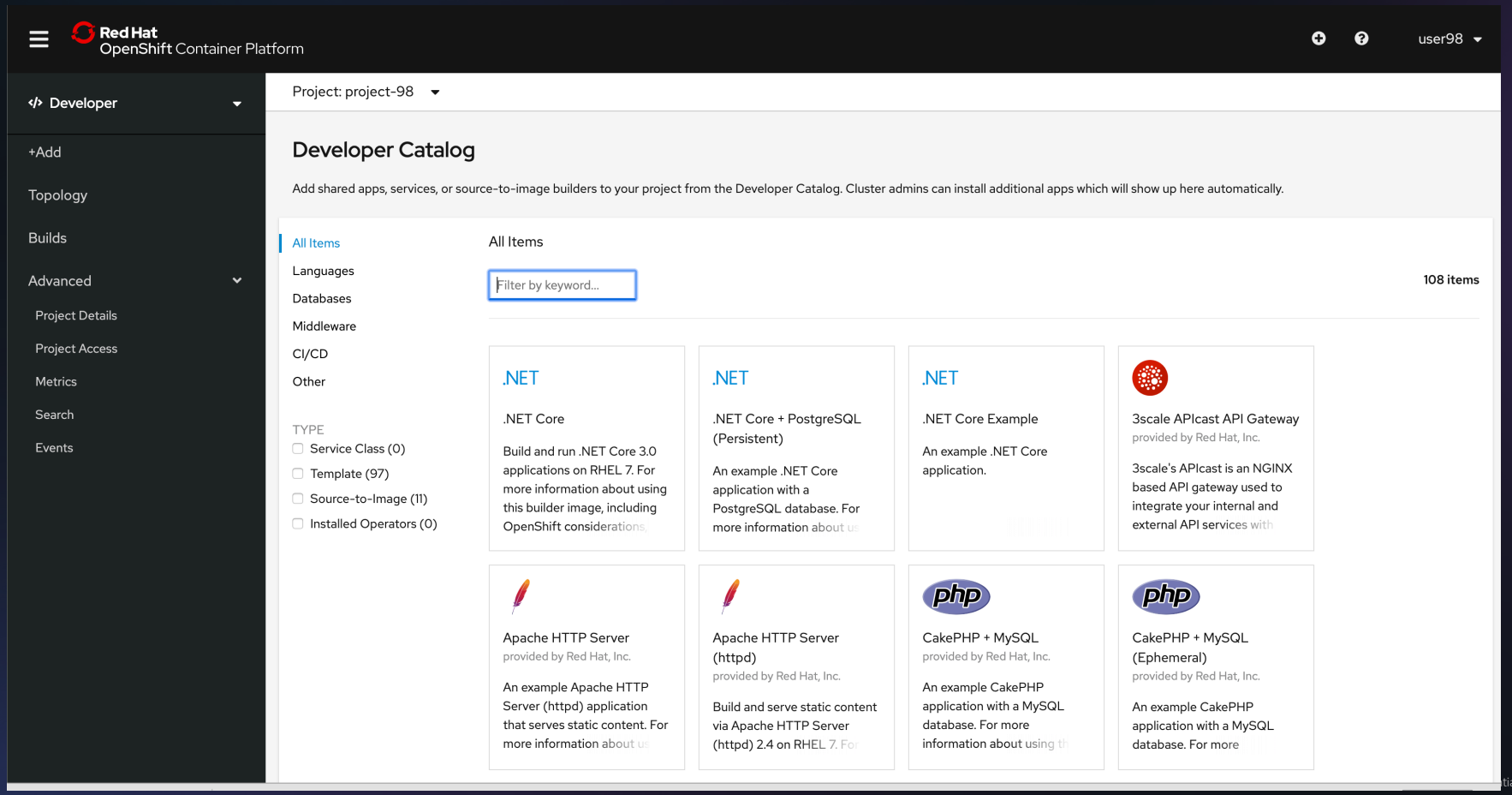
# Openshift UI Navigation



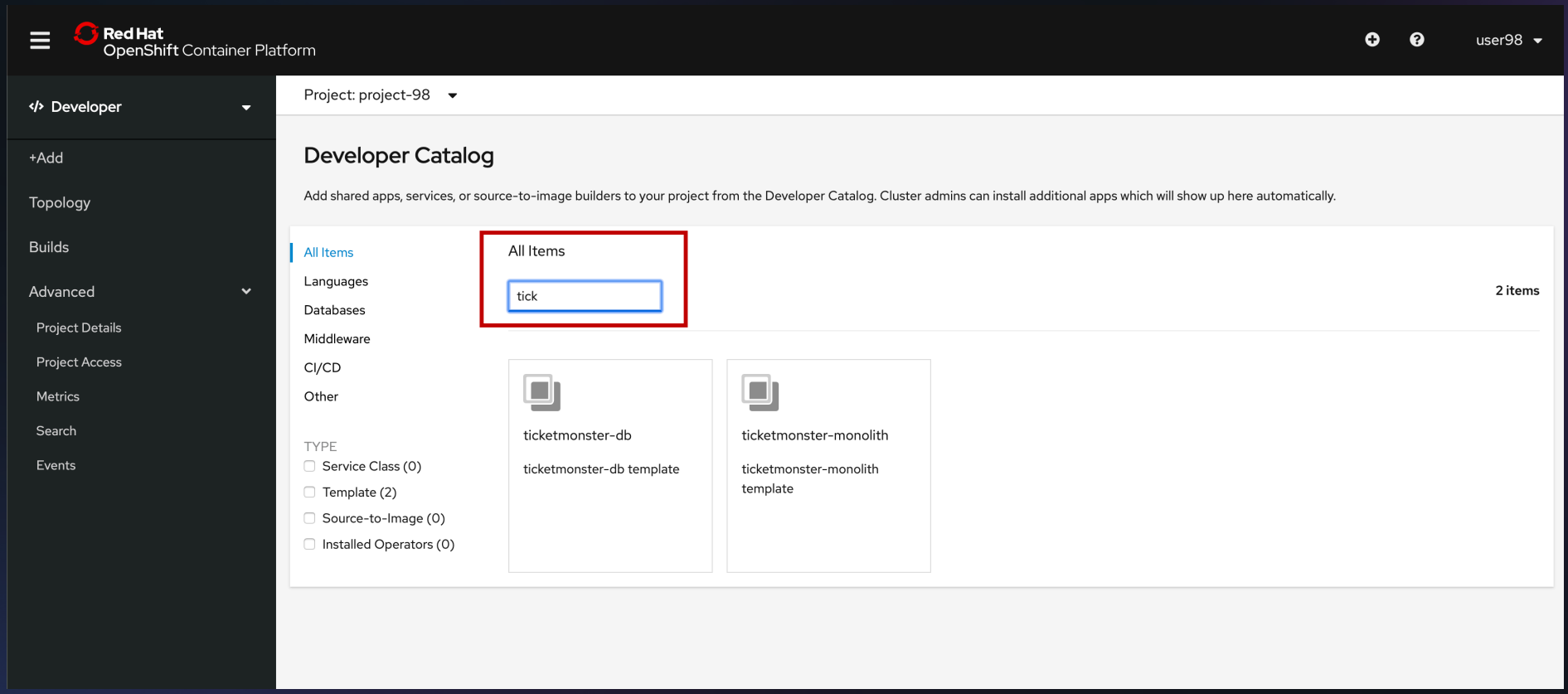
# Openshift UI Navigation



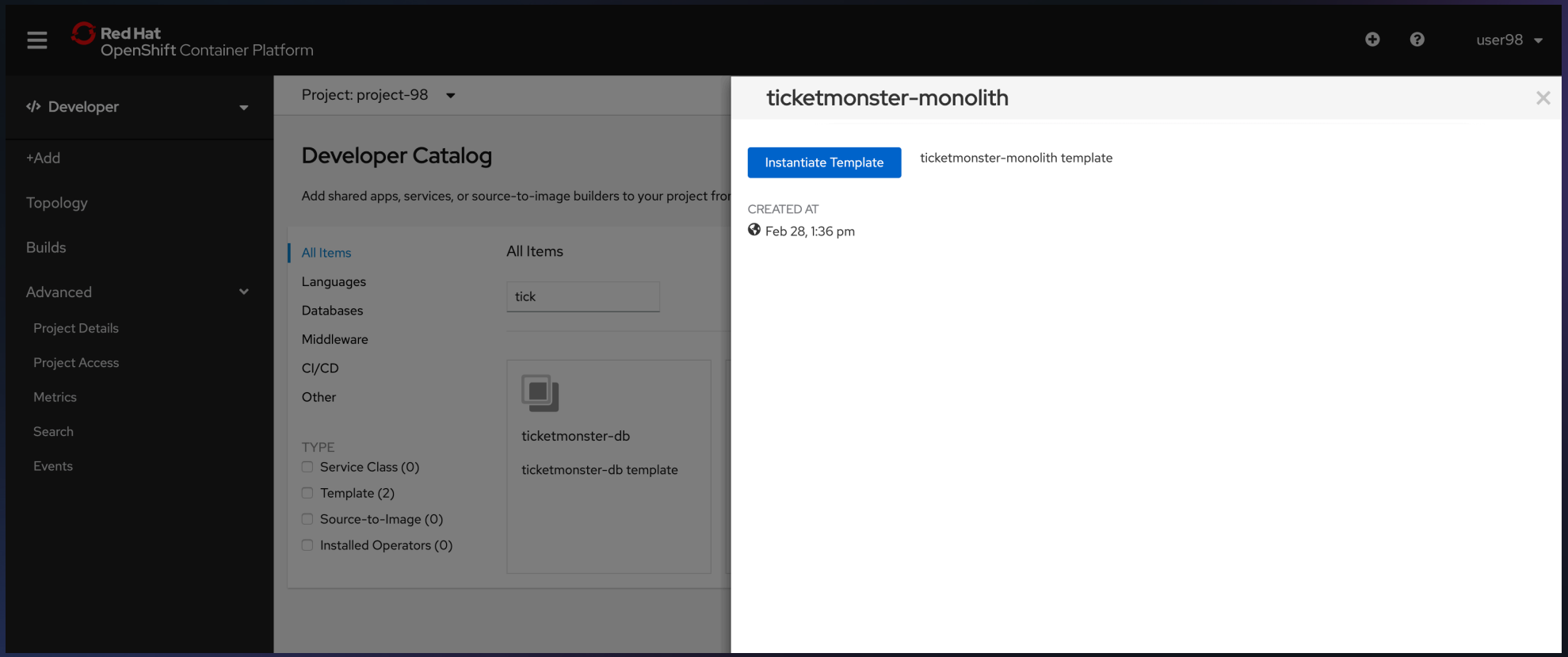
# Openshift UI Navigation



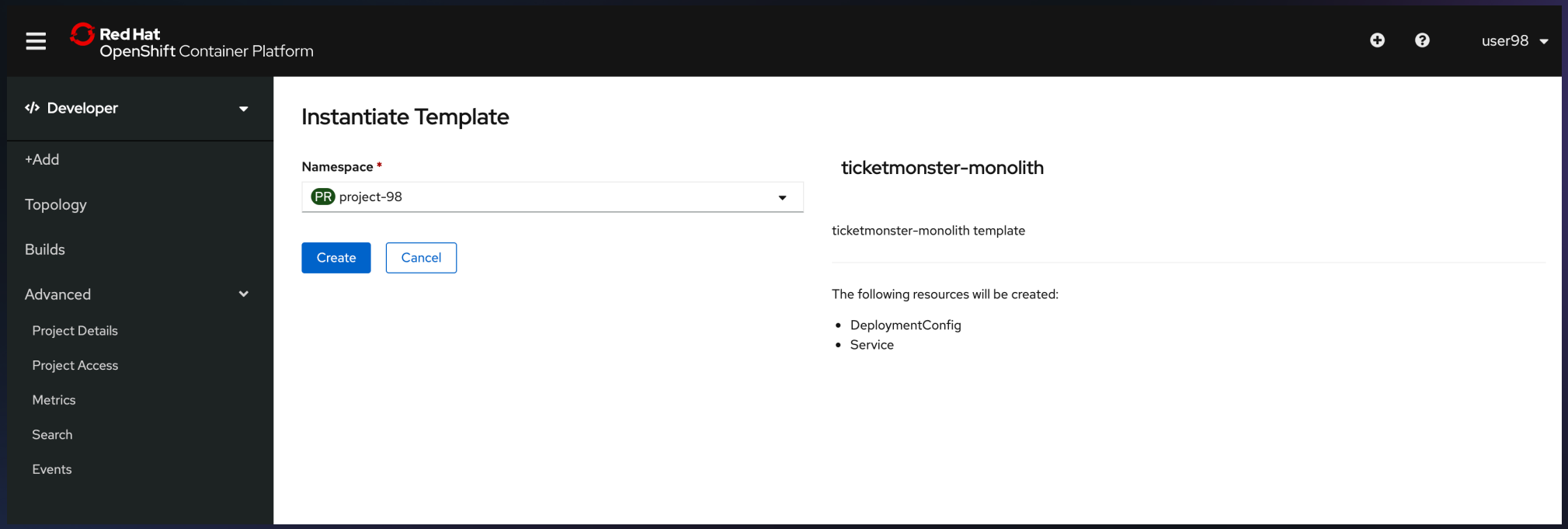
# Openshift UI Navigation



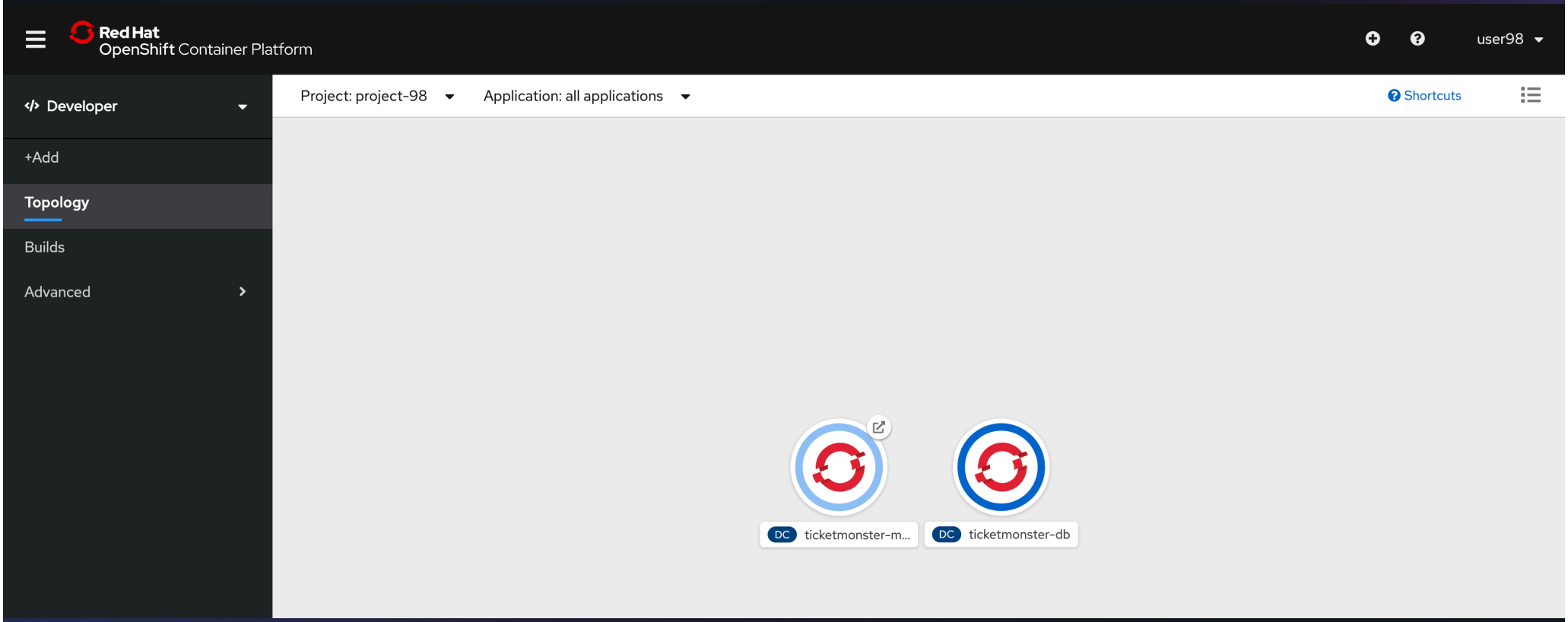
# Openshift UI Navigation



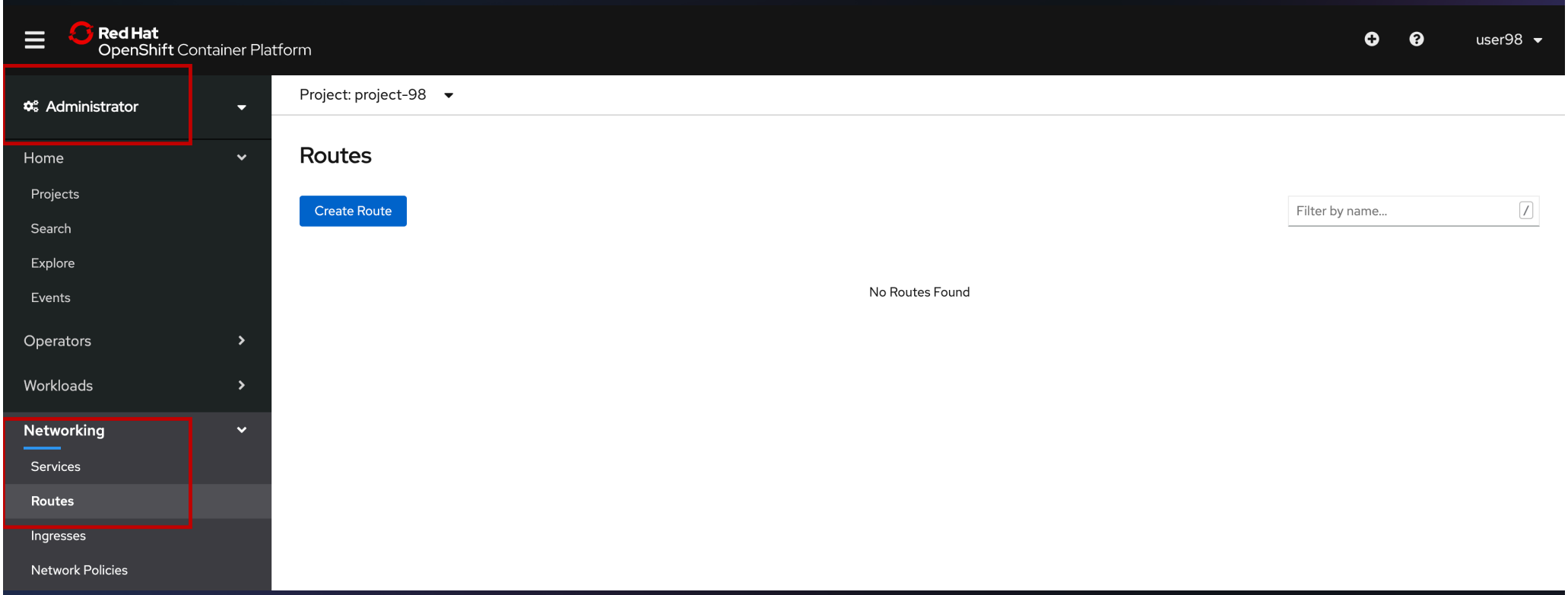
# Openshift UI Navigation



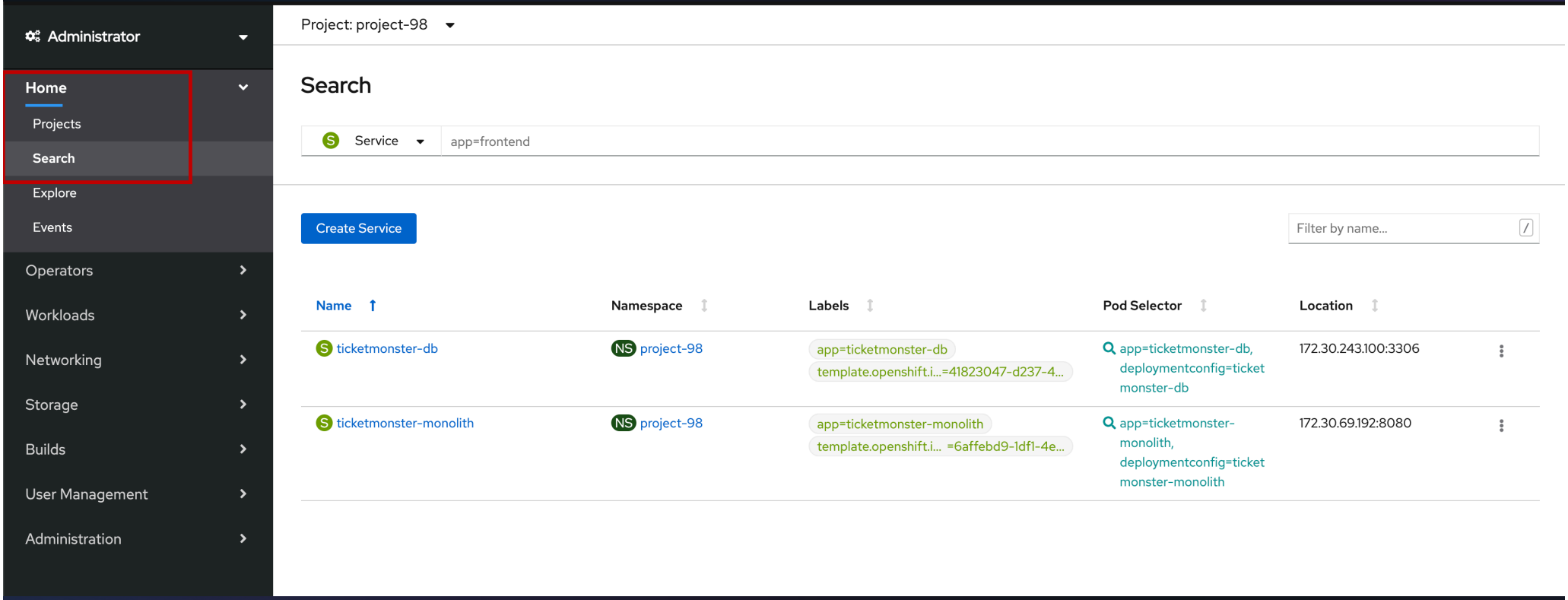
# Openshift UI Navigation



# Openshift UI Navigation

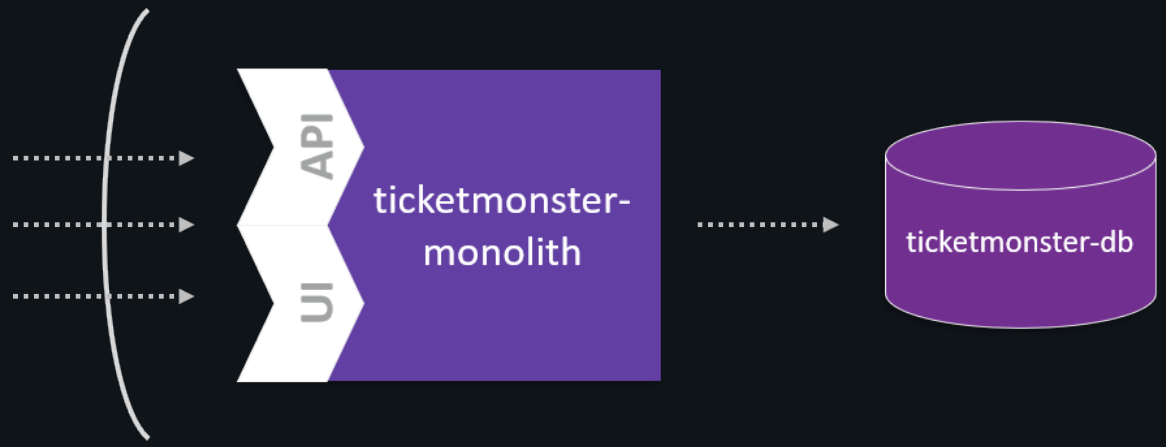


# Openshift UI Navigation



## Lab: Lift-and-shift TicketMonster

- Overview
  1. Create a MySQL service for monolith
  2. Push application to OpenShift
  3. Bind MySQL service to monolith




- Instructions:
  - Lab: 1_Lift-and-Shift_TicketMonster

- Takeaways
  - Familiarization with OpenShift

## Step 1:

## Deploy Database for Monolith

a. Click on “Add”

b. Import YAML/JSON

c. Navigate to git repo for ticketmonster-db template

d. Copy RAW YAML

e. Create db template

f. Instantiate db template

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/1_Lift-and-Shift_TicketMonster

## Step 2:

## Deploy Monolith

a. Click on “Add”

b. Import YAML/JSON

c. Navigate to git repo for ticketmonster-monolith-template

d. Copy RAW YAML

e. Create monolith template

f. Instantiate monolith template

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/1_Lift-and-Shift_TicketMonster

## Step 3:

## Create Route to Monolith

a. Click on “Administrator”

b. Networking -> Routes -> Create Route

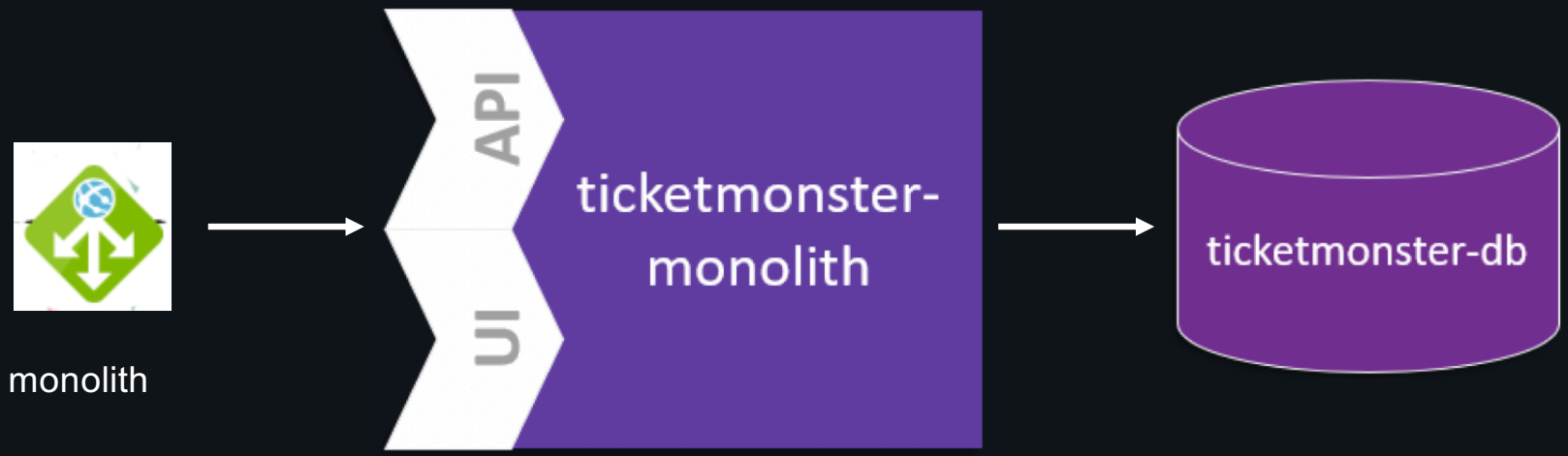
c. Name = monolith

d. Service = ticketmonster-monolith

e. Target Port = 8080 -> 8080 (TCP)

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/1_Lift-and-Shift_TicketMonster

## Lab: Lift-and-shift TicketMonster



## Step 4:

## Exercise the application

a. In Route, copy “Location”

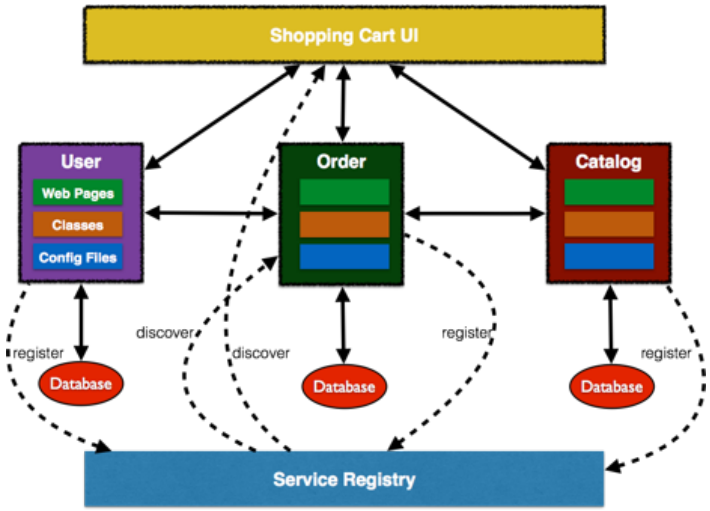
b. Paste the location url in a browser and exercise the application

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/1_Lift-and-Shift_TicketMonster

## Extract the UI from the Monolith

A single application consolidating all client interactions for the monolith:

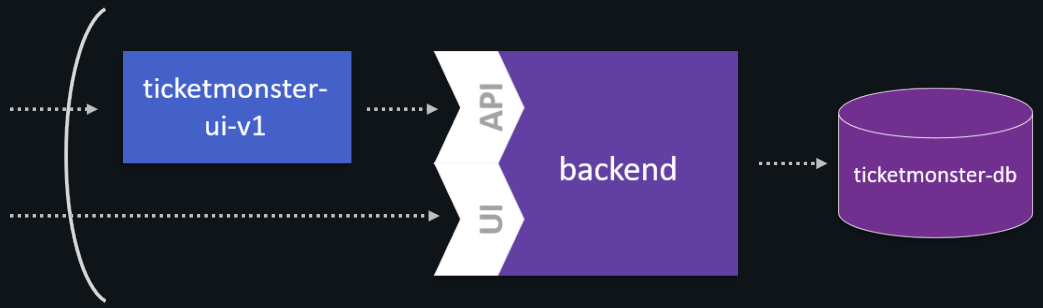
- Application uses the services provided by the monolith and compose them together
  - Use an API gateway or service mesh when application needs to invoke calls to several microservices, topic of separate workshop
- Works as proxy where the UI pages of different components are invoked to show the interface



## Lab: Extract the UI from the Monolith

- Overview
  1) Rename ‘monolith‘ to ‘backend‘
  2) Switch to tm-ui-vi
  3) Check Proxy and ReverseProxy
  4) Push new UI to OpenShift
- Instructions:
  - Lab: 2_Extract_UI_From_Monolith
- Takeaways
  - Deployment of Microservice

To start breaking up the monolith, a best practice is extracting the user interface from TicketMonster since this helps to decouple the client facing part from the business logic. So, this lab launches the first microservice to separate the user interface from the rest of TicketMonster as depicted below.




for instructions with OC CLI, you can use this link: http://bit.ly/m2m-step2

## Step 1:

## Define a new route for the monolith

a. Click on “Administrator”

b. Networking -> Routes -> Create Route

c. Name = backend

d. Service = ticketmonster-monolith

e. Target Port = 8080 -> 8080 (TCP)

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/1_Lift-and-Shift_TicketMonster

**Step 1:**

**Define a new route for the monolith**

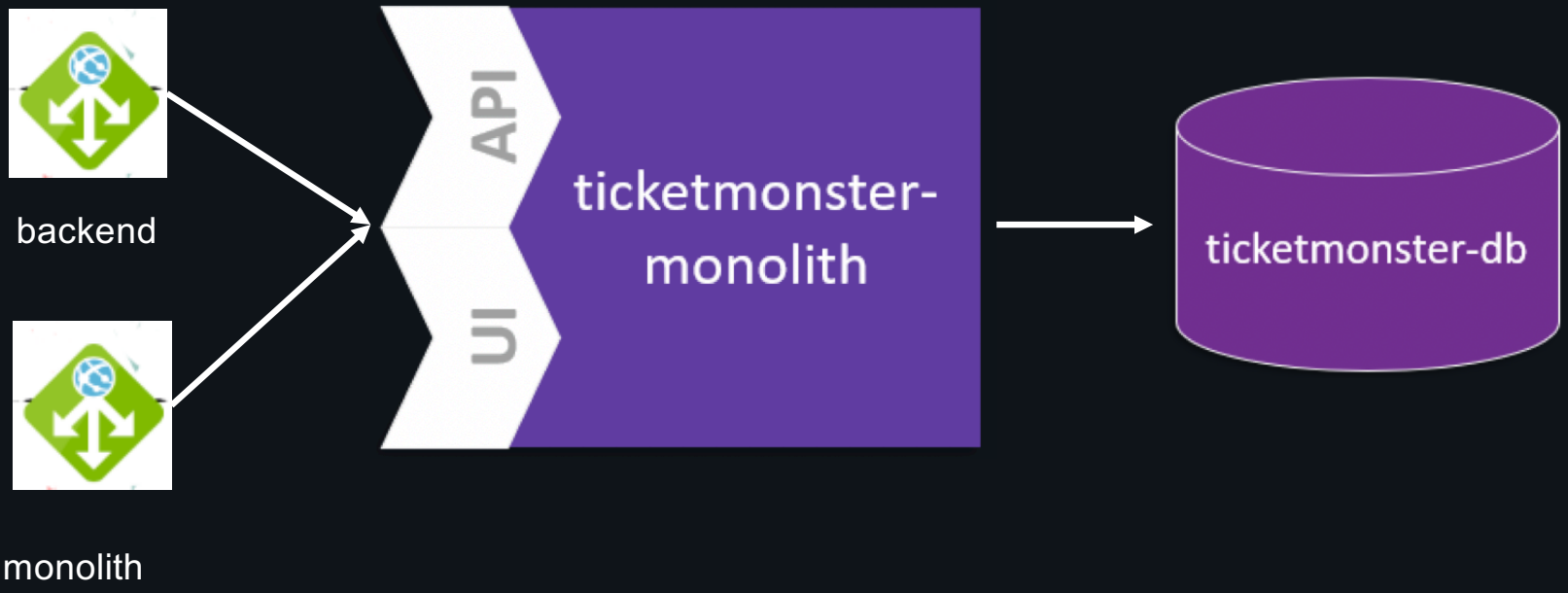
# Save backend route “host” info in notepad

# e.g

backend-project-98.apps.cluster-dallas-f761.dallas-f761.example.opentlc.com

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/1_Lift-and-Shift_TicketMonster

## Lab: Lift-and-shift TicketMonster



## Step 2:

## Decouple UI from the Monolith

a. Click on “Add”

b. Import YAML/JSON

c. Navigate to git repo for ticketmonster-ui-v1 template

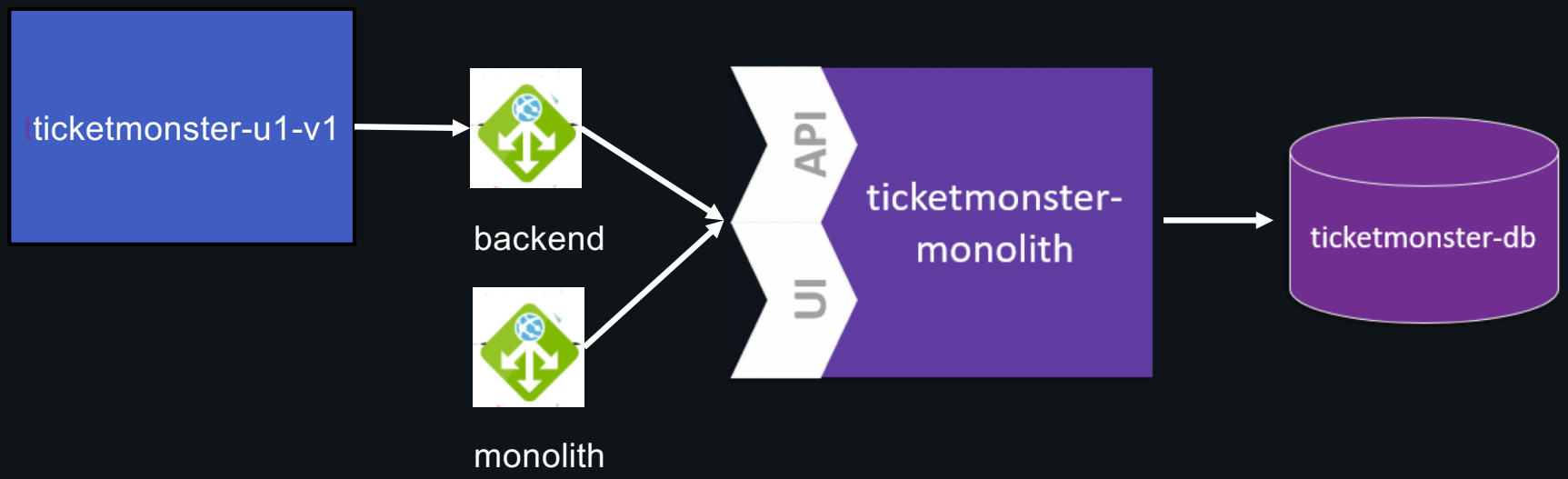
d. Paste RAW YAML into OpenShift

e. Create monolith template

f. During instantiate step, provide backend URL (without https://).

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/2_Extract_UI_From_Monolith

## Lab: Lift-and-shift TicketMonster



## Step 1:

## Define a new route for the monolith

a. Click on “Administrator”

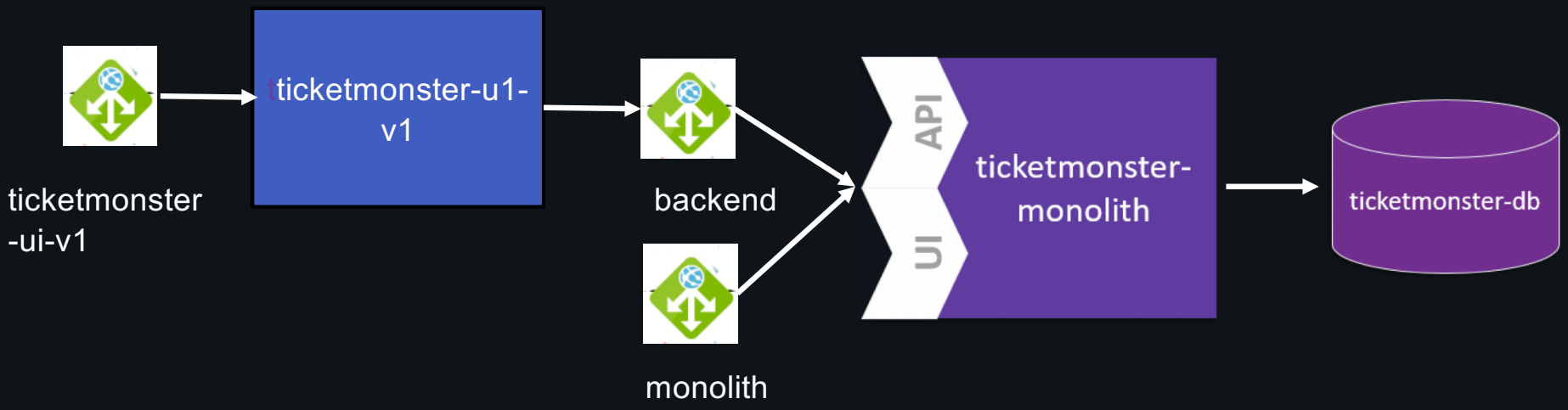
b. Networking -> Routes -> Create Route

c. Name = ticketmonster-ui-v1

d. Service = ticketmonster-ui-v1

e. Target Port = 8080 -> 8080 (TCP)

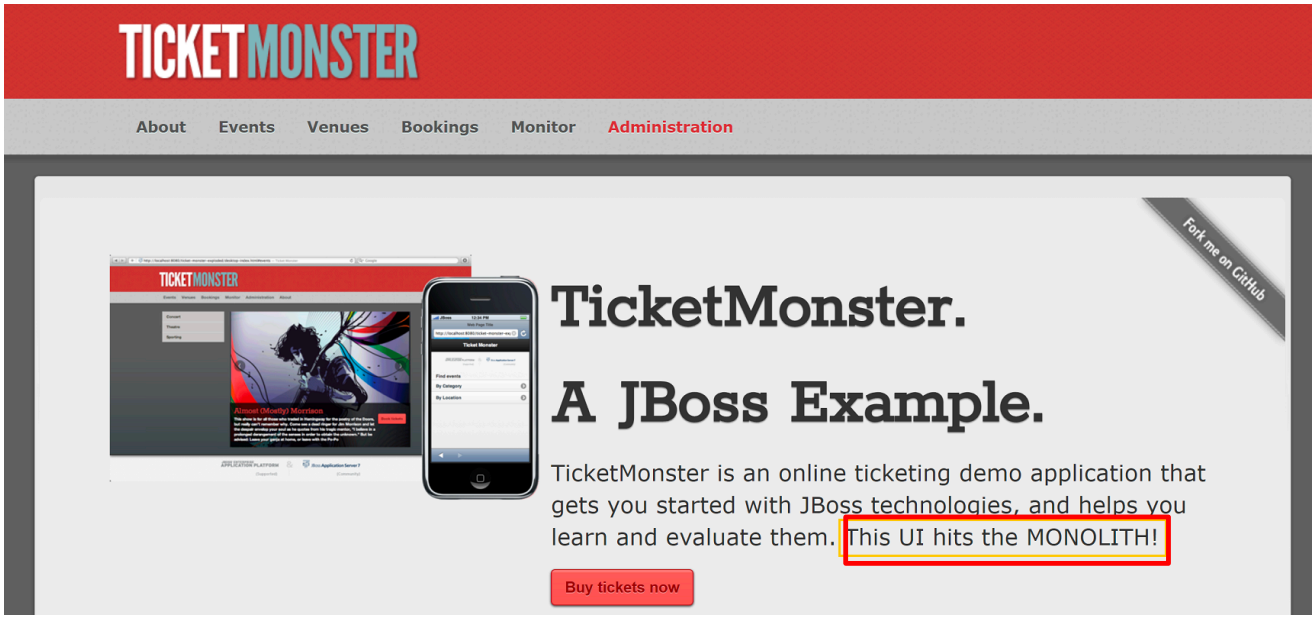
## Lab: Lift-and-shift TicketMonster



## Step 3:

## Validate new UI

a. Select “ticketmonster-UI” URL from the route section and validate that the page states “This UI hits the MONOLITH”




https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/2_Extract_UI_From_Monolith

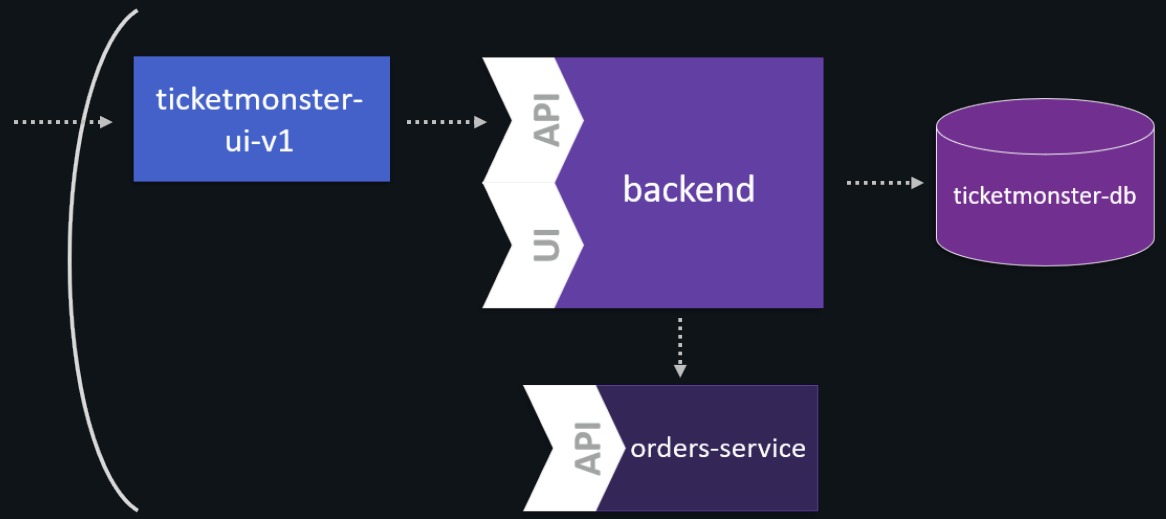
## Identify a Microservice

- **Problem**: we don’t know much about our monolith:
  - Who is depending on us and how are they depending on us?
  - Whom are we depending on and how are we depending on them?
  - What happens within our monolith code base when it gets called?

- Leverage Dynatrace to:
  - Get Dependency Information
  - Detect Service Endpoints, Usage & Behavior
  - Understand Service Flow per Endpoint
  - Finding Entry Points with CPU Sampling Data
  - Define Custom Service Entry Points

## Lab: Identify a Microservice - Monitoring as part of the platform

- Overview
  1) Analyze UI and monolith
  2) Define Custom Service Detection Rule
- Instructions:
  - Lab: 4_Identify_a_Microservice
- Takeaways
  - Virtually broken the monolith



## Identify the Domain Model of the Microservice

- Data management of a decoupled Microservice:
  - Use an existing API of the monolith to keep data management at the monolith
  - (If no appropriate API is available) create a new lower-level API for the microservice (but still keep the data management at the monolith)
  - Do an ETL (Extract, Transform, Load) from the monolith's database to the microservice' database and keep both in sync

- **Data model** shows how data is stored and entities relate to each other in the persistence layer.
- **Domain model** describes the *behavior* of the solution space of a microservice's domain and tends to focus on use case scenarios

**Step 1:**

**Define custom service entry point**

a. **Log into Dynatrace Tenant**

b. **Go to Settings, Server-side service monitoring, and click on Custom service detection”**

c. **Click on Define Java service, set name of custom service to “project-xx-orders-service” and click Find entry point**

d. **Select the process group that contains your entry point “project-xx-ticketmonster-monolith” and click Continue.**

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/4_Identify_a_Microservice

## Step 2:

## Define custom service entry point

a. Search for loaded classes and interfaces with name BookingService, select “org.jboss.examples.ticketmonster.rest.BookingService” and click Continue.

b. Use Selected Class and click Continue

c. Select method with “createBooking” as entry point and click Finish

d. Review and select ”Save Changes”

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/4_Identify_a_Microservice

## Step 3:

## Restart Pod to activate custom service

a. In Openshift UI, Select Administrator

b. Workloads -> Pods

c. Select Pod “ticketmonster-monolith-xxxx”

d. Action -> Delete Pod

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/4_Identify_a_Microservice

## Step 4:

## Book a ticket on Ticket Monster

a. Open the ”ticketmonster-ui-v1” in a browser

b. Click on Events, Concerts, and on, eg. Rock concert of the decade

c. Select Venue, Date, and Order Ticket

d. Select section and number of tickets

e. Checkout and review booking details

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/4_Identify_a_Microservice

## Step 5:

## Consider Service Flow in Dynatrace

a. Log into Dynatrace console and select user ## management zone

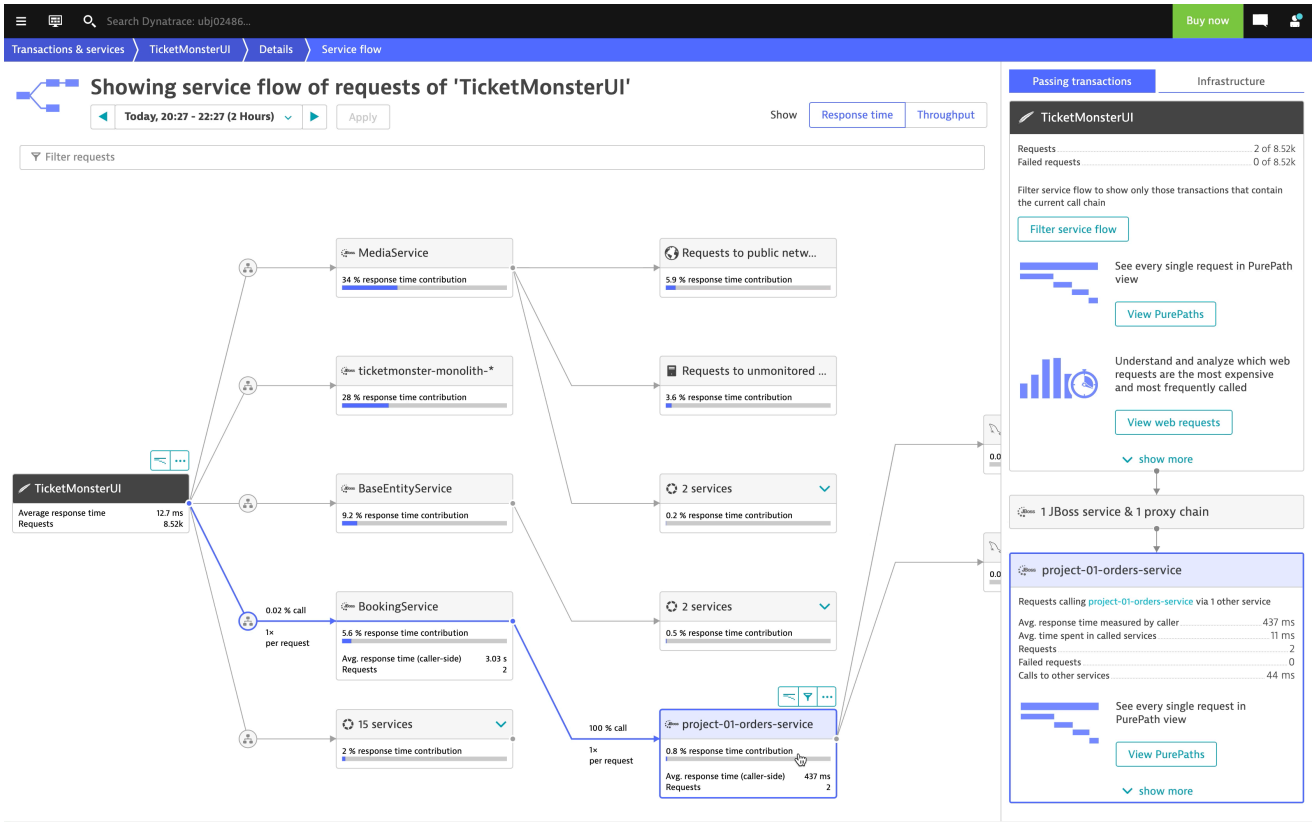
b. Choose the Transactions & services tab from left menu

c. Select TicketMonsterUI and click on View service flow

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/4_Identify_a_Microservice

# Step 5:

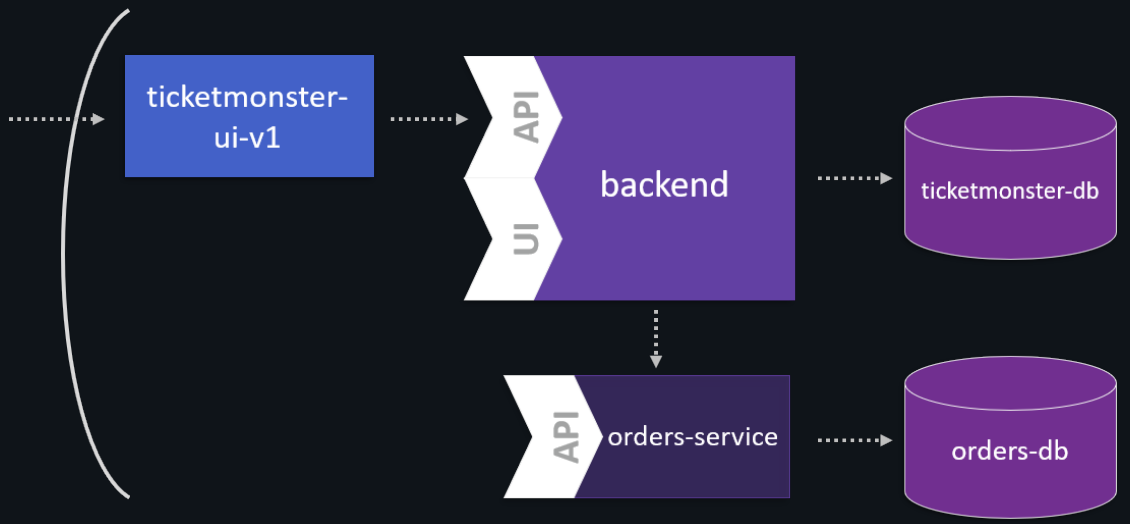
# Consider Service Flow in Dynatrace




https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/4_Identify_a_Microservice

## Lab: The Microservice and its Domain Model

- Overview
  1. Analyze database queries of Microservice
- Instructions:
  - Lab: 5_Domain_Model_of_Microservice
- Takeaways
  - Persistence layer of Microservices



## Refactor your Source Code

- 1) Extract OrdersService from the Monolith
  - Cut out the class containing createBooking method
  - Identify missing abstractions for a successful build
- 2) Strangle the OrdersService around the Monolith
  - Call the OrdersService in the business logic of the backend
- 3) Build the Domain Model for the Microservice
  - Integrate a data virtualization framework into your code base
  - Extend the data model to the domain model by virtualized data views

## Step 1:

## Inspect the Domain Model with Dynatrace

a. **Within Dynatrace Service-flow, follow through custom service to mysql db**

b. **Click on “View database statements”**

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/4_Identify_a_Microservice

## Step 2:

## Create new database for Orders microservice

a. Select “Add”

b. Import YAML/JSON

c. Navigate to git repo for ticketmonster orders-db template

d. Paste RAW YAML into OpenShift

e. Create and Instantiate

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/5_Domain_Model_of_Microservice

**Step 3:**

**Setup the database**

a. **Administrator -> Workloads -> Pod**

b. **Select the “orders-db” pod**

c. **Select the “Terminal” tab**

d. **In the terminal type:**

```
a.cd~
b.pwd
```

***Should now be in directory /var/lib/mysql**

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/5_Domain_Model_of_Microservice

## Step 3:

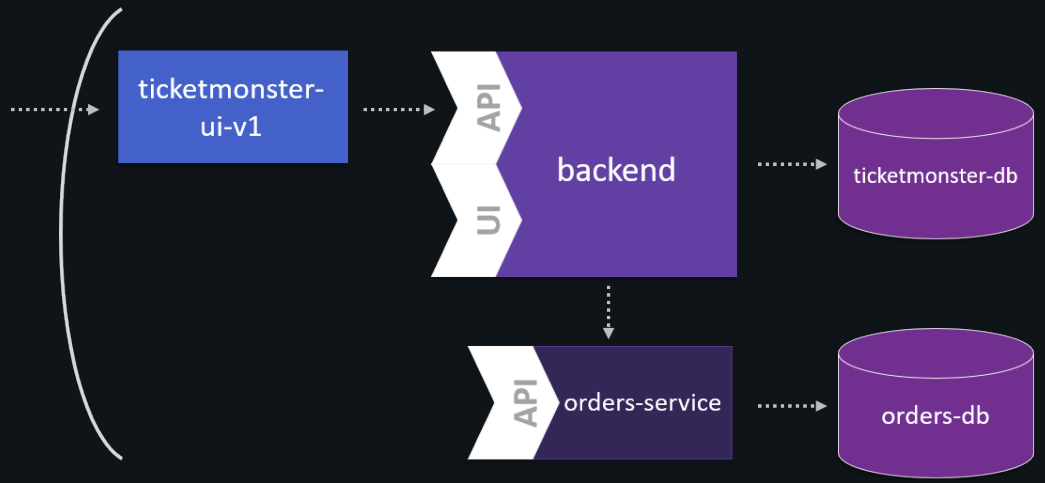
## Setup the database

```
curl -o 0_ordersdb-schema.sql https://raw.githubusercontent.com/dynatrace-innovationlab/monolith-to-microservice-openshift/master/orders-service/src/main/resources/db/migration/0_ordersdb-schema.sql

curl -o 1_ordersdb-data.sql https://raw.githubusercontent.com/dynatrace-innovationlab/monolith-to-microservice-openshift/master/orders-service/src/main/resources/db/migration/1_ordersdb-data.sql

mysql -u root orders < 0_ordersdb-schema.sql

mysql -u root orders < 1_ordersdb-data.sql

mysql -u root

GRANT ALL ON orders.* TO 'ticket'@'%';

show databases;

use orders

show tables;

select * from id_generator;
```

## Lab: Deploy the Microservice

- Overview
  1. Deploy a new backend version
  2. Create a MySQL service instance for microservice
  3. Build microservice and push it to OpenShift
  4. Bind MySQL service instance to microservice
  5. Re-deploy UI to use new backend version
- Instructions:
  - Lab: 6_Deploy_the_Microservice
- Takeaways
  - Microservice for a specific bounded context

Based on the result of the previous labs, you identified the microservice **OrdersService** that has its own code repository and defines its own domain model. To launch this service, it is not recommended to directly route traffic to this new service since we cannot fully ensure that it works as supposed. For this reason, we strangle the microservice around the monolith. In other words, all incoming requests will still be intercepted by the backend service, which forwards synthetic or live traffic to **OrdersService**.
In this lab you'll use feature flags and OpenShift routing mechanism to smoothly incorporate the new microservice into the monolith. The final state of this lab is shown below:



## Step 1:

## Deploy the Orders microservice

a. **Developer -> Add**

b. **Navigate to git repo for ticketmonster-orders-service template**

c. **Copy RAW YAML**

d. **Paste RAW YAML into OpenShift**

e. **Create and Instantiate**

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/6_Deploy_the_Microservice

## Step 2:

## Create route for the orders microservice

a. Administrator -> Networking - Routes

b. Select “Create Route” for ticketmonster-orders-service

c. Name: orders-service

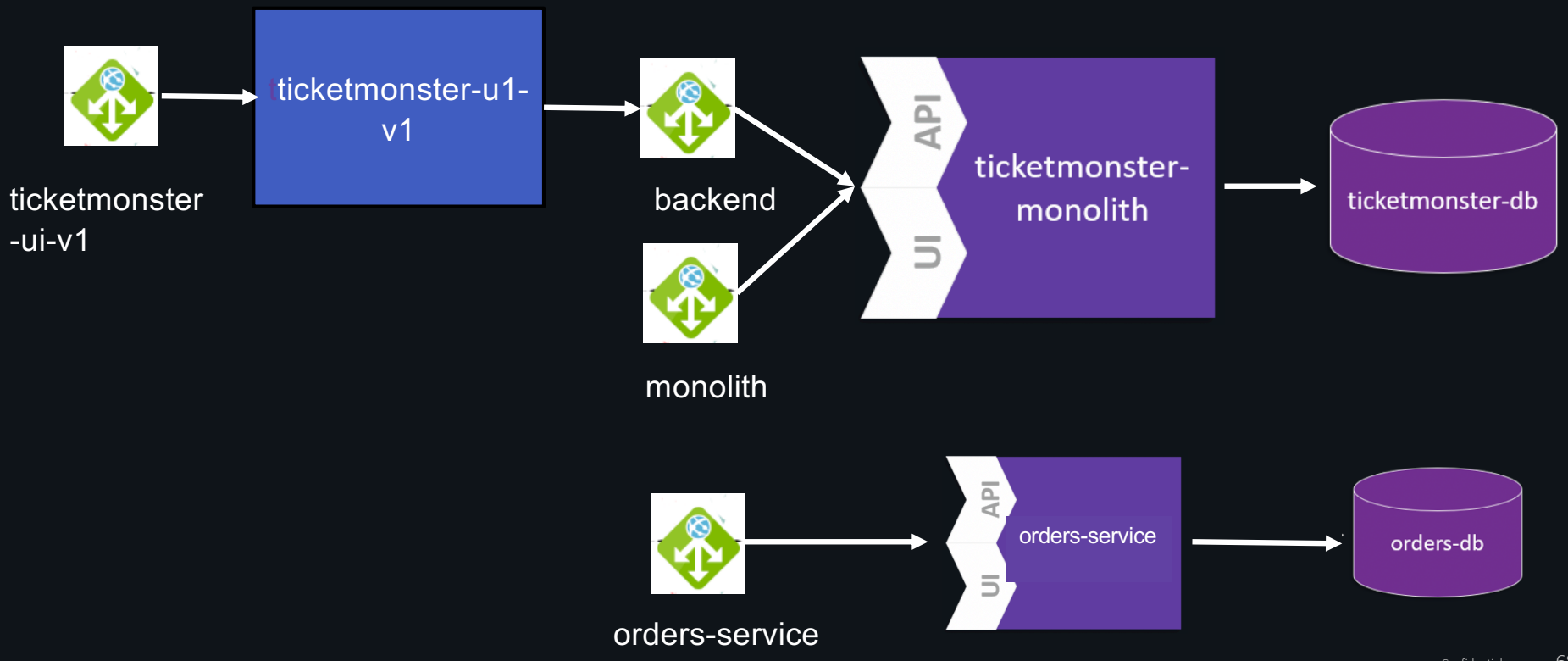
d. Service: ticketmonster-orders-service

e. Target Port: 8080 -> 8080 (TCP)

f. Select “Create”

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/6_Deploy_the_Microservice

## Lab: Identify the microservice



## Step 3:

## Deploy ticketmonster-backend-V2

a. **Developer -> Add**

b. **Navigate to git repo for ticketmonster-backend-v2 template**

c. **Copy RAW YAML**

d. **Paste RAW YAML into OpenShift**

e. **Create and Instantiate**

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/6_Deploy_the_Microservice

## Step 4:

## Create route for ticketmonster-backend-V2

a. Administrator -> Networking - Routes

b. Select “Create Route” for ticketmonster-backend-v2

c. Name: backend-v2

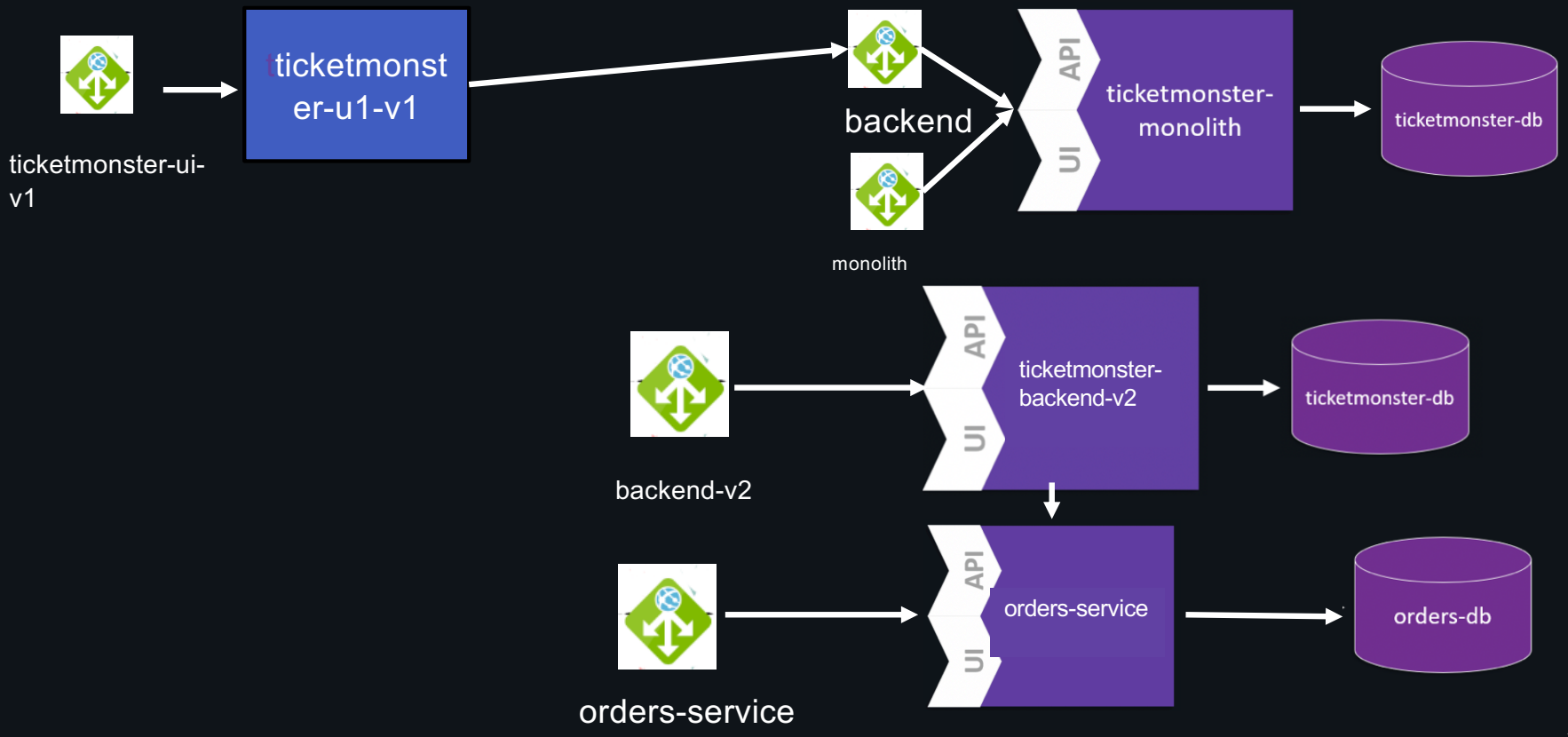
d. Service: ticketmonster-backend-v2

e. Target Port: 8080 -> 8080 (TCP)

f. Select “Create”

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/6_Deploy_the_Microservice

## Lab: Identify the microservice



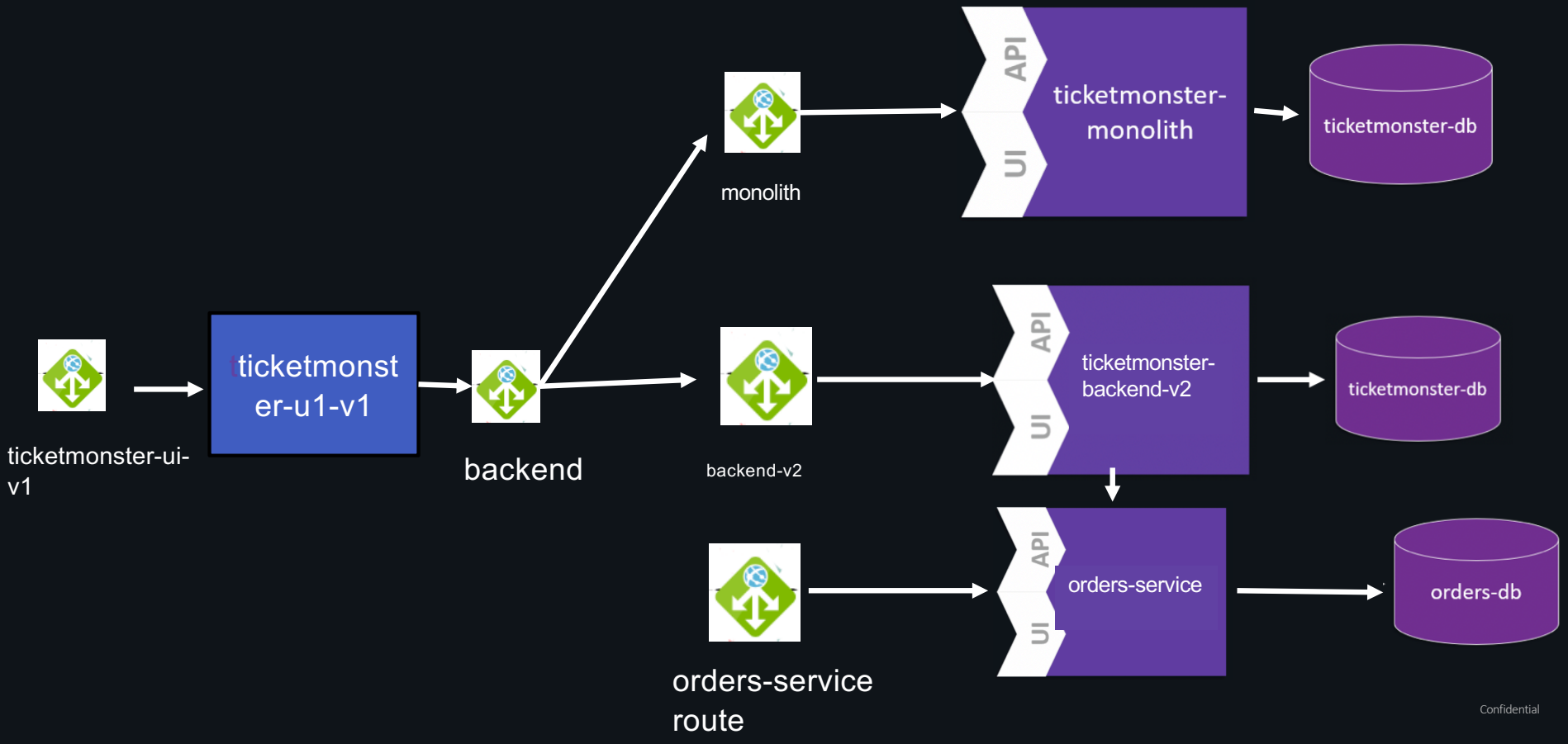
## Step 5:

## Split route to backend

### a. Use OC command to split the load to the backend

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/6_Deploy_the_Microservice

## Lab: Identify the microservice



## Step 6:

## Verify Service in Dynatrace

**We can verify the service flow in Dynatrace.**

**From the left menu, choose the Transaction & services tab.**

**Select service TicketMonsterUI.**

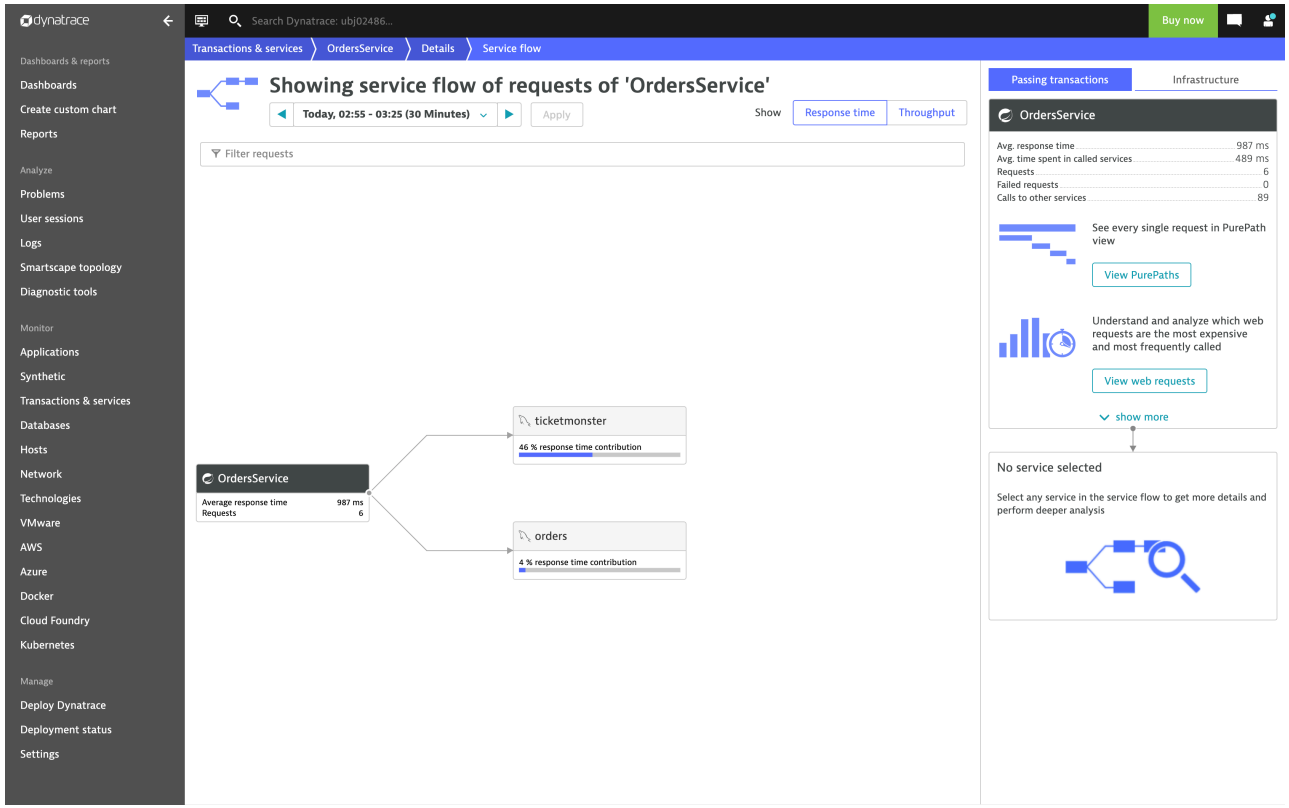
**Click on View service flow.**

**Finally, you see the service flow containing the microservice orders-service.**

https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/6_Deploy_the_Microservice

# Step 6:

# Verify Service in Dynatrace




https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/6_Deploy_the_Microservice