

Step 1: Setup Project

- Login to OpenShift
- Create Project-##
- Navigate to Project

The screenshot shows the Red Hat OpenShift Container Platform interface. On the left, there's a sidebar with 'Developer' selected. In the center, under 'Topology', it says 'No projects exist' and 'Select one of the following options'. A modal window titled 'Create Project' is open, prompting for 'Name *' (with an empty input field), 'Display Name' (empty), and 'Description' (empty). At the bottom of the modal are 'Cancel' and 'Create' buttons. Below the modal, there are three options: 'From Catalog' (with a book icon), 'Create resources from their YAML or JSON definitions' (with a gear icon), and 'Browse the catalog to discover database services to add to your application' (with a database icon).

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

Openshift UI Navigation

The screenshot shows the OpenShift Container Platform interface. At the top, there is a dark header bar with the Red Hat logo and the text "OpenShift Container Platform". On the right side of the header are three small icons and the text "user98". Below the header is a navigation bar with a dropdown menu. The dropdown menu has three items: "Developer" (which is highlighted with a red border), "Administrator", and "Developer". To the left of the navigation bar, there are links for "Builds" and "Advanced". The main content area is titled "Add" and contains a message "No workloads found" followed by a sub-instruction "To add content to your project, create an application, component or service using one of these options.". Below this message are five cards, each representing a way to add a workload:

- From Git**: Import code from your git repository to be built and deployed.
- Container Image**: Deploy an existing image from an image registry or image stream tag.
- From Catalog**: Browse the catalog to discover, deploy and connect to services.
- From Dockerfile**: Import your Dockerfile from your git repo to be built & deployed.
- YAML**: Create resources from their YAML or JSON definitions.

At the bottom of the page, there is some very small, partially visible text: "idential 24".

Openshift UI Navigation

The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text "Red Hat OpenShift Container Platform", and user information for "user98". The left sidebar has a "Developer" section with a dropdown menu, a "+Add" button (which is currently selected), and three other options: "Topology", "Builds", and "Advanced". The main content area is titled "Add" and displays a message: "No workloads found" followed by "To add content to your project, create an application, component or service using one of these options.". Below this are five cards representing different deployment methods:

- From Git**: Import code from your git repository to be built and deployed.
- Container Image**: Deploy an existing image from an image registry or image stream tag.
- From Catalog**: Browse the catalog to discover, deploy and connect to services.
- From Dockerfile**: Import your Dockerfile from your git repo to be built & deployed.
- YAML**: Create resources from their YAML or JSON definitions. This card is highlighted with a red border.

At the bottom of the main content area, there is a "Database" card with the text: "Browse the catalog to discover database services to add to your application".

Openshift UI Navigation

The screenshot shows the Red Hat OpenShift Container Platform interface. At the top, there's a dark header bar with the Red Hat logo and the text "Red Hat OpenShift Container Platform". Below the header, the navigation bar indicates "Project: project-98" and "Application: all applications". On the far right of the header, there are user profile icons for "user98".

The main content area is titled "Add" and displays a message: "No workloads found" followed by "To add content to your project, create an application, component or service using one of these options.". Below this message, there are five options:

- From Git**: Import code from your git repository to be built and deployed.
- Container Image**: Deploy an existing image from an image registry or image stream tag.
- From Catalog**: (highlighted with a red border) Browse the catalog to discover, deploy and connect to services.
- From Dockerfile**: Import your Dockerfile from your git repo to be built & deployed.
- YAML**: Create resources from their YAML or JSON definitions.

At the bottom left, there's a separate box labeled "Database" with the sub-instruction: "Browse the catalog to discover database services to add to your application".

Openshift UI Navigation

The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text "Red Hat OpenShift Container Platform", and user information for "user98". The left sidebar has a "Developer" section with a dropdown menu, followed by links for "+Add", "Topology", "Builds", "Advanced" (with a dropdown menu for "Project Details", "Project Access", "Metrics", "Search", and "Events"), and a "TYPE" filter section with checkboxes for "Service Class (0)", "Template (97)", "Source-to-Image (1)", and "Installed Operators (0)". The main content area is titled "Developer Catalog" and displays a list of items under ".NET", "Apache HTTP Server", and "php". Each item card includes a thumbnail, the application name, a brief description, and a "View Details" button.

Project: project-98

Developer Catalog

Add shared apps, services, or source-to-image builders to your project from the Developer Catalog. Cluster admins can install additional apps which will show up here automatically.

All Items

Filter by keyword...

108 items

.NET

.NET Core
Build and run .NET Core 3.0 applications on RHEL 7. For more information about using this builder image, including OpenShift considerations.

.NET Core + PostgreSQL (Persistent)
An example .NET Core application with a PostgreSQL database. For more information about using this builder image, including OpenShift considerations.

.NET Core Example
An example .NET Core application.

3scale APIcast API Gateway
provided by Red Hat, Inc.
3scale's APIcast is an NGINX based API gateway used to integrate your internal and external API services with

Apache HTTP Server
provided by Red Hat, Inc.
An example Apache HTTP Server (httpd) application that serves static content. For more information about using this builder image, including OpenShift considerations.

Apache HTTP Server (httpd)
provided by Red Hat, Inc.
Build and serve static content via Apache HTTP Server (httpd) 2.4 on RHEL 7. For more information about using this builder image, including OpenShift considerations.

php

CakePHP + MySQL (Ephemeral)
provided by Red Hat, Inc.
An example CakePHP application with a MySQL database. For more information about using this builder image, including OpenShift considerations.

php

CakePHP + MySQL
provided by Red Hat, Inc.
An example CakePHP application with a MySQL database. For more information about using this builder image, including OpenShift considerations.

Openshift UI Navigation

The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, 'OpenShift Container Platform', a user icon for 'user98', and a dropdown menu. The left sidebar, titled 'Developer', contains links for '+Add', 'Topology', 'Builds', 'Advanced' (with 'Project Details', 'Project Access', 'Metrics', 'Search', and 'Events' sub-links), and 'Events'. The main content area is titled 'Developer Catalog' and displays the message: 'Add shared apps, services, or source-to-image builders to your project from the Developer Catalog. Cluster admins can install additional apps which will show up here automatically.' A red box highlights the 'All Items' link under the 'Languages' section. Below this, there are two cards: one for 'ticketmonster-db' and one for 'ticketmonster-monolith'. Each card has a thumbnail, the app name, and a 'template' link. On the far right, it says '2 items'. At the bottom of the sidebar, there's a 'TYPE' section with checkboxes for 'Service Class (0)', 'Template (2)', 'Source-to-Image (0)', and 'Installed Operators (0)'.

Project: project-98

Developer Catalog

Add shared apps, services, or source-to-image builders to your project from the Developer Catalog. Cluster admins can install additional apps which will show up here automatically.

All Items

Languages

Databases

Middleware

Other

ticketmonster-db

ticketmonster-db template

ticketmonster-monolith

ticketmonster-monolith template

2 items

TYPE

Service Class (0)

Template (2)

Source-to-Image (0)

Installed Operators (0)

Openshift UI Navigation

The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, 'OpenShift Container Platform', and user information ('user98'). The left sidebar has a 'Developer' section selected, with options like '+Add', 'Topology', 'Builds', 'Advanced' (expanded), 'Project Details', 'Project Access', 'Metrics', 'Search', and 'Events'. The main content area shows a 'Developer Catalog' for 'Project: project-98'. It features a search bar, a 'ticketmonster-monolith' template card, and a 'ticketmonster-db' template card. The catalog also includes sections for 'All Items' (Languages, Databases, Middleware, CI/CD, Other) and 'TYPE' filters (Service Class, Template, Source-to-Image, Installed Operators).

ticketmonster-monolith

Instantiate Template ticketmonster-monolith template

CREATED AT Feb 28, 1:36 pm

ticketmonster-db

ticketmonster-db template

All Items

Languages

Databases

Middleware

CI/CD

Other

TYPE

Service Class (0)

Template (2)

Source-to-Image (0)

Installed Operators (0)

Openshift UI Navigation

The screenshot shows the OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the text "OpenShift Container Platform", and a user dropdown for "user98". The left sidebar has a "Developer" section selected, with options like "+Add", "Topology", "Builds", "Advanced" (expanded to show "Project Details", "Project Access", "Metrics", "Search", and "Events"), and a "Create" button. The main content area is titled "Instantiate Template" and shows a "Namespace" dropdown set to "PR project-98". To the right, the template "ticketmonster-monolith" is listed, along with its description: "ticketmonster-monolith template". It also lists the resources that will be created: "DeploymentConfig" and "Service".

Openshift UI Navigation

The screenshot shows the OpenShift Container Platform UI interface. At the top, there is a dark header bar with the Red Hat logo and the text "Red Hat OpenShift Container Platform". On the right side of the header, there are icons for adding a new item, help, and user authentication ("user98"). Below the header is a navigation sidebar on the left. The sidebar has a "Developer" section with a dropdown menu set to "Project: project-98" and "Application: all applications". Other options in the sidebar include "+Add", "Builds", and "Advanced". The main content area is titled "Topology". It displays two application icons: "ticketmonster-m..." and "ticketmonster-db", each represented by a blue and red circular icon with a white arrow. Below each icon is a small "DC" badge. The bottom right corner of the main content area contains the text "Confidential" and the number "31".

Openshift UI Navigation

The screenshot shows the OpenShift Container Platform UI. The top bar includes the Red Hat logo, the project dropdown "Project: project-98", and user information "user98". The left sidebar has a "Administrator" section with "Home", "Projects", "Search", "Explore", and "Events". Below it is a "Networking" section with "Services", "Routes" (which is selected and highlighted with a red box), and "Ingresses" and "Network Policies". The main content area shows the "Routes" page for the "project-98" project. It features a "Create Route" button and a "Filter by name..." input field. A message "No Routes Found" is displayed.

Red Hat
OpenShift Container Platform

Administrator

Home

Projects

Search

Explore

Events

Operators

Workloads

Networking

- Services
- Routes**
- Ingresses
- Network Policies

Project: project-98

Create Route

Filter by name...

No Routes Found

Openshift UI Navigation

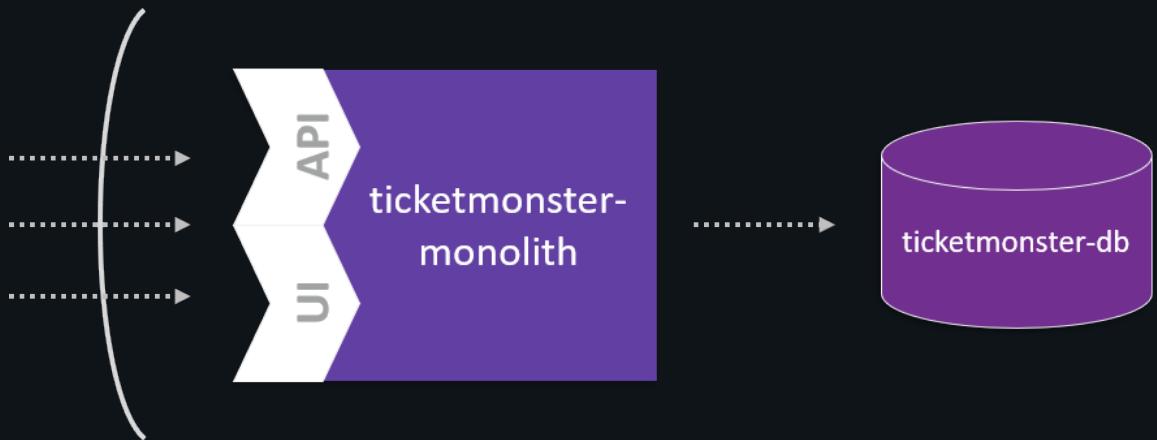
The screenshot shows the OpenShift UI navigation bar with the project set to 'project-98'. The 'Search' menu item is highlighted with a red box. The main search interface displays a list of services, with two entries visible:

Name	Namespace	Labels	Pod Selector	Location
ticketmonster-db	project-98	app=ticketmonster-db template.openshift.i...=41823047-d237-4...	app=ticketmonster-db, deploymentconfig=ticket monster-db	172.30.243.100:3306
ticketmonster-monolith	project-98	app=ticketmonster-monolith template.openshift.i...=6affebd9-1df1-4e...	app=ticketmonster- monolith, deploymentconfig=ticket monster-monolith	172.30.69.192:8080



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:

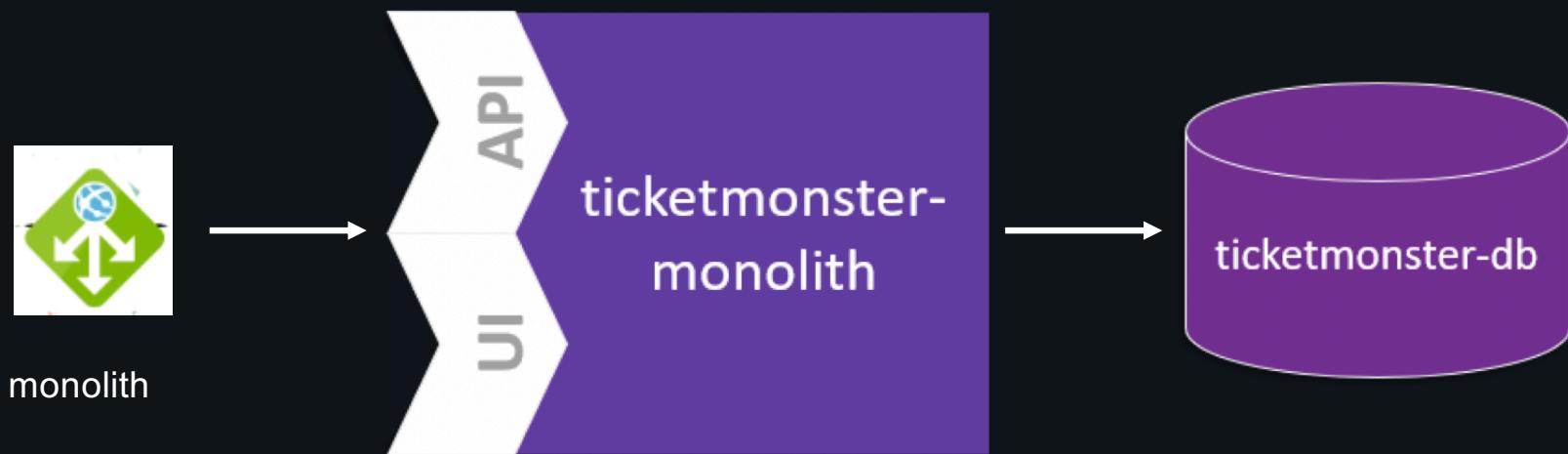
Create Route to Monolith

- 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

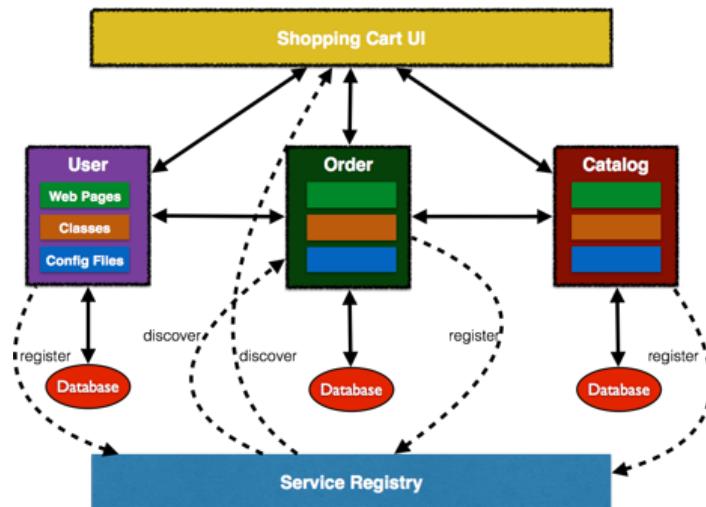
https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/1_Lift-and-Shift_TicketMonster



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

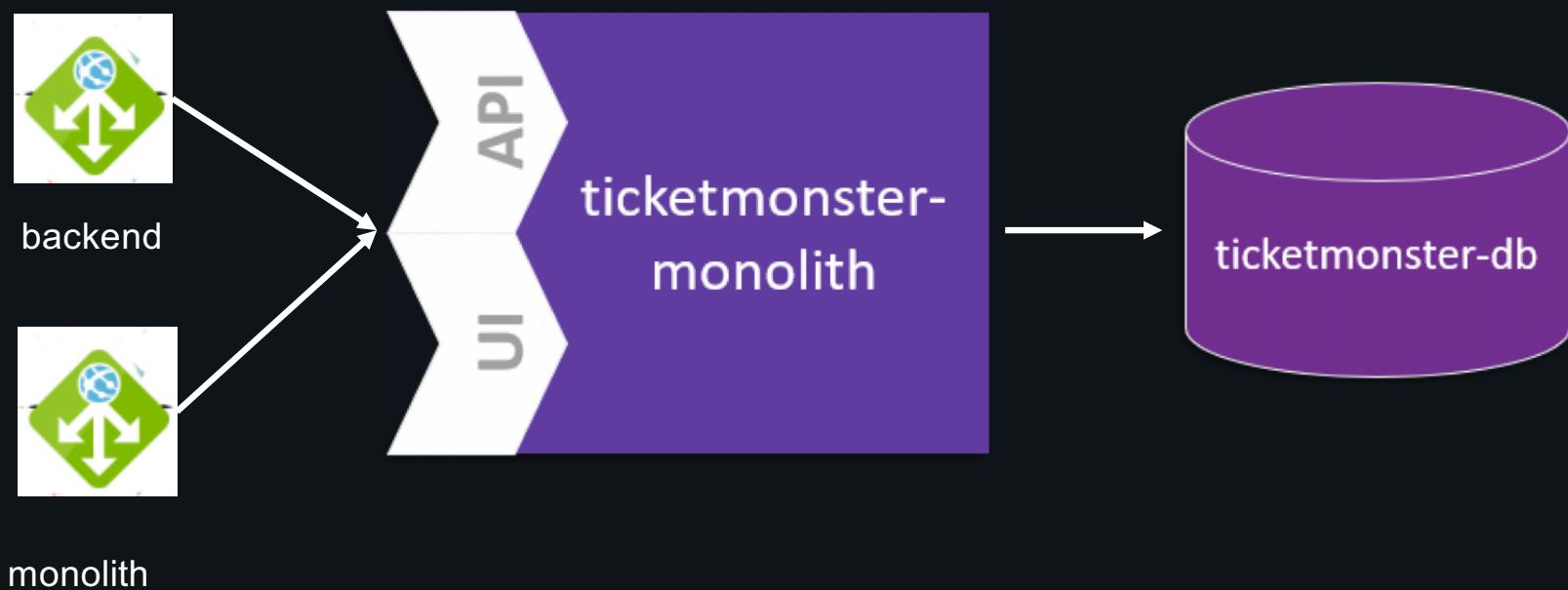
https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/1_Lift-and-Shift_TicketMonster

Confidential

43



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

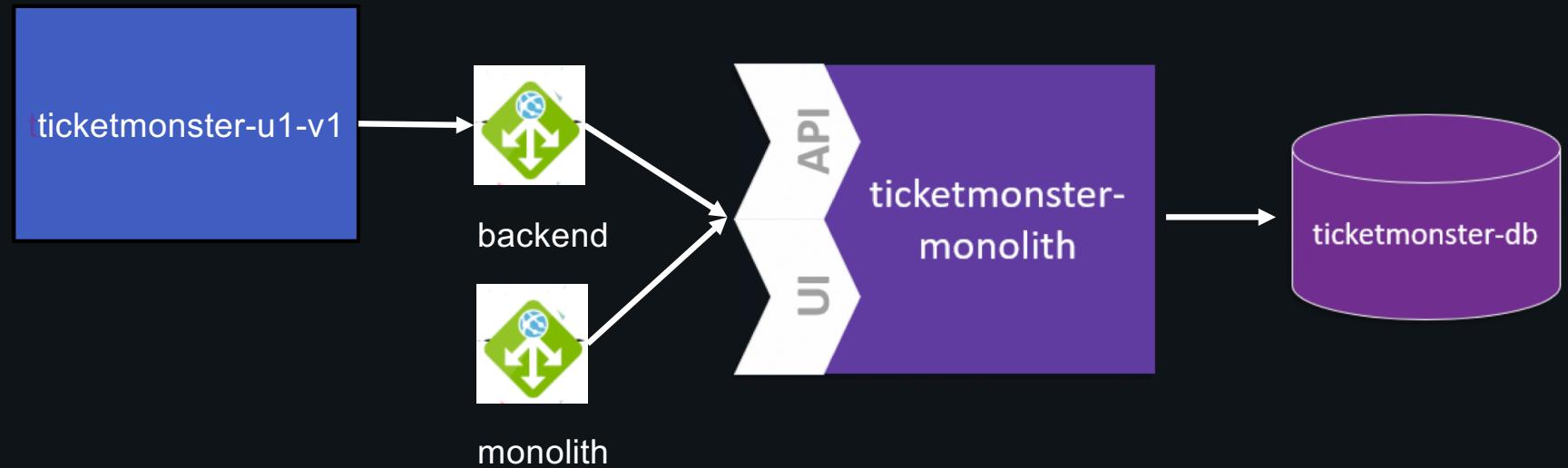
- e. Create monolith template

- 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





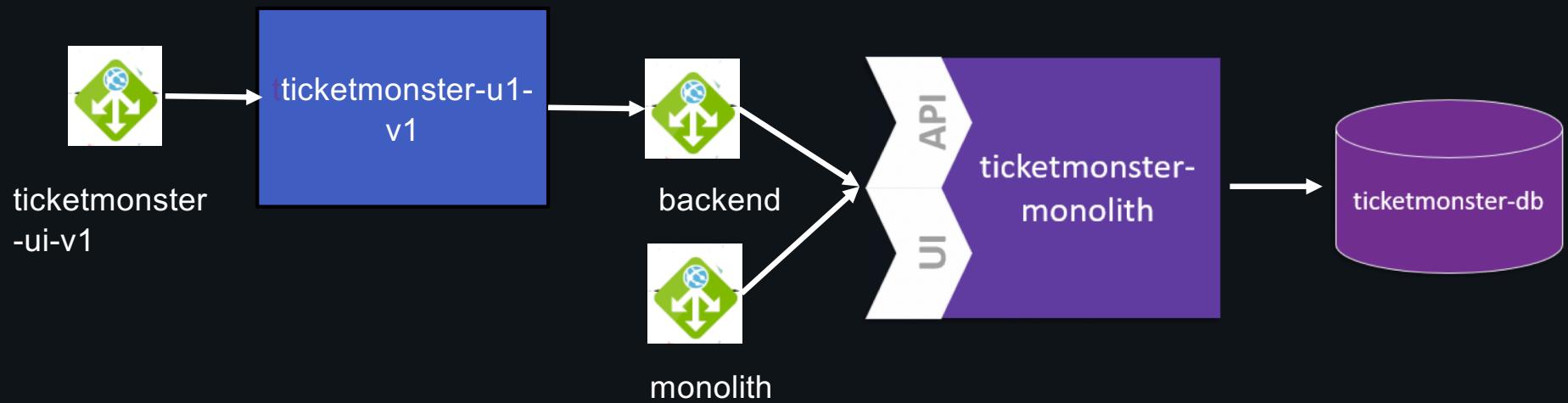
Step 1:

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”

The screenshot shows the homepage of the TicketMonster application. At the top, there is a red header bar with the "TICKETMONSTER" logo. Below the header is a grey navigation bar with links: About, Events, Venues, Bookings, Monitor, and Administration. The main content area features a large image of a person performing on stage, with the text "Almost Ready! Morris" overlaid. To the left of the main image, there are two smaller images: a desktop screen showing a ticket listing and a smartphone displaying a ticket detail page. To the right, there is a dark grey callout box containing the text "Fork me on GitHub". The central heading reads "TicketMonster. A JBoss Example." Below the heading, a paragraph describes the application as an online ticketing demo that helps users learn JBoss technologies. A red rectangular box highlights the text "This UI hits the MONOLITH!". At the bottom of the page is a red button labeled "Buy tickets now".

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



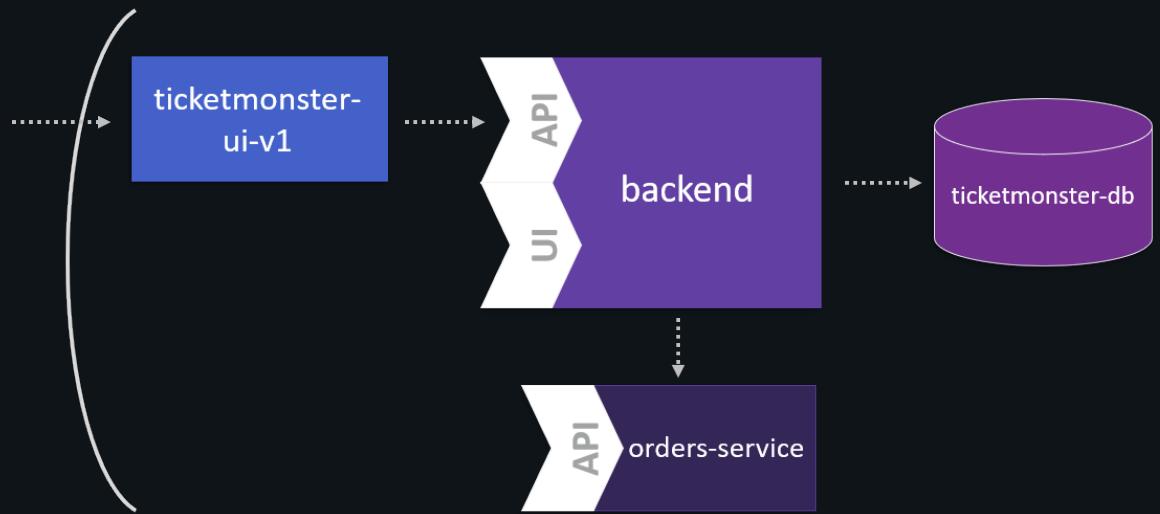
Identify a Microservice

- **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

Confidential

53



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_Identifier_a_Microservice

Confidential

54



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

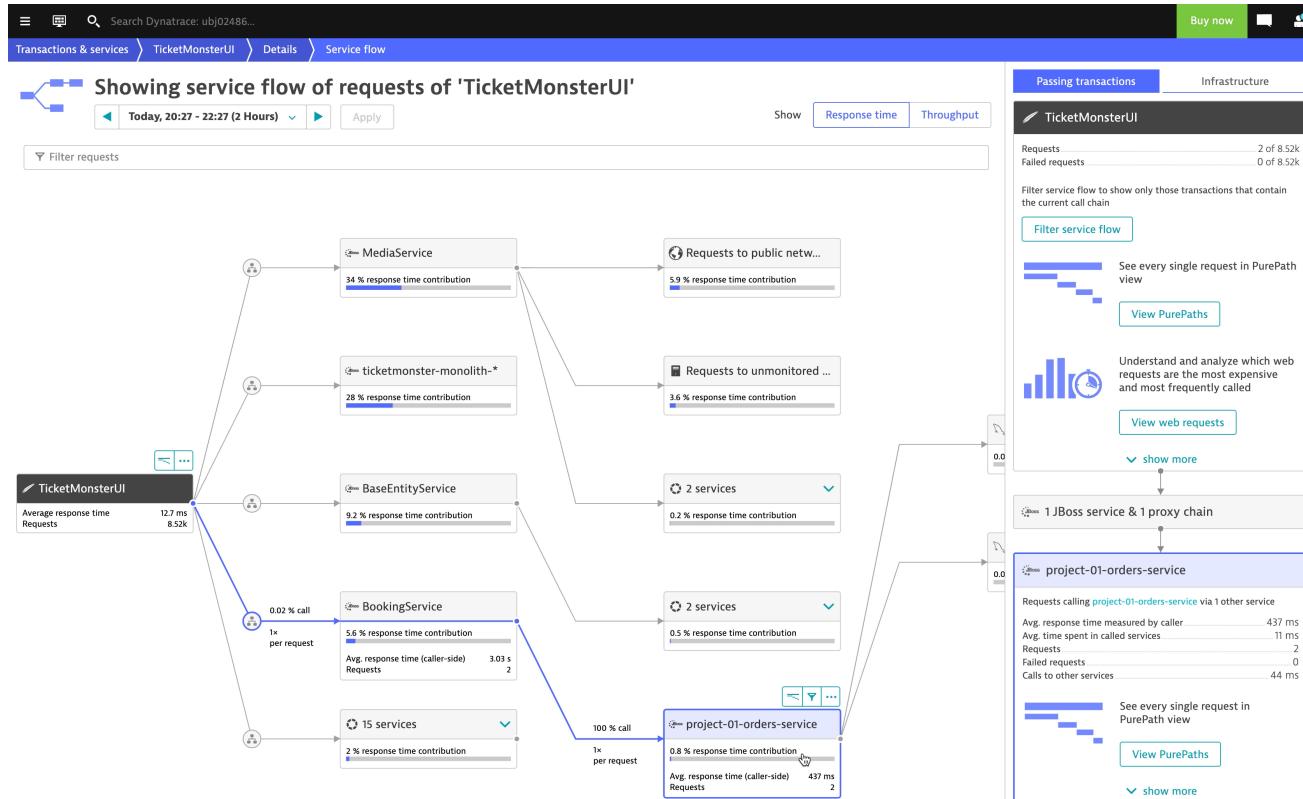
- a. Log into Dynatrace console and select user ## management zone
- b. Choose the Transactions & services tab from left menu
- c. Select TicketMonsterUI and click on View service flow

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



Step 5:

Consider Service Flow in Dynatrace

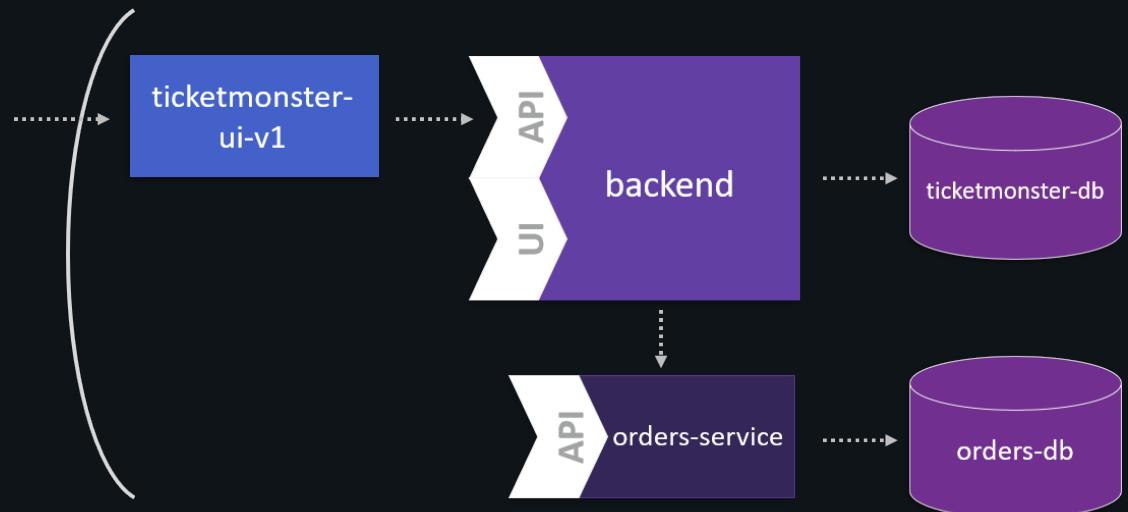


https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/4_Identifier_a_Microservice



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;
```

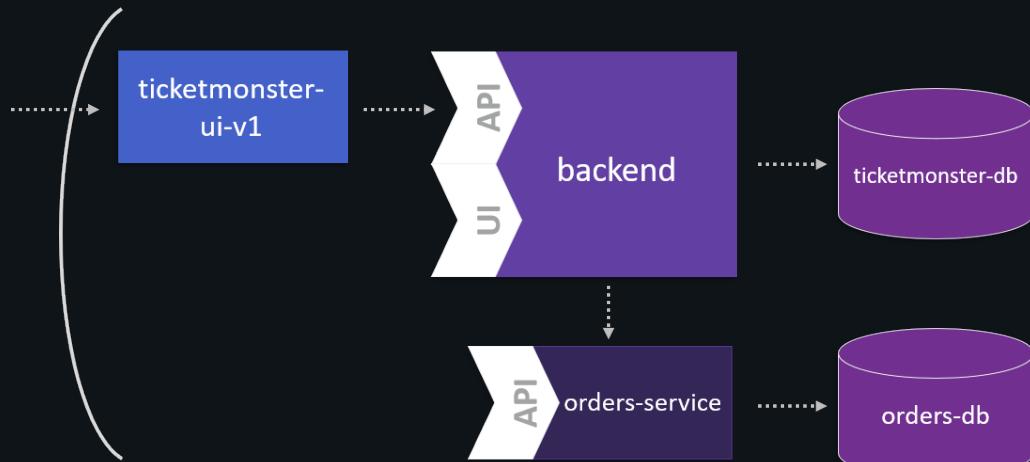


Lab: Deploy the Microservice

- 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:

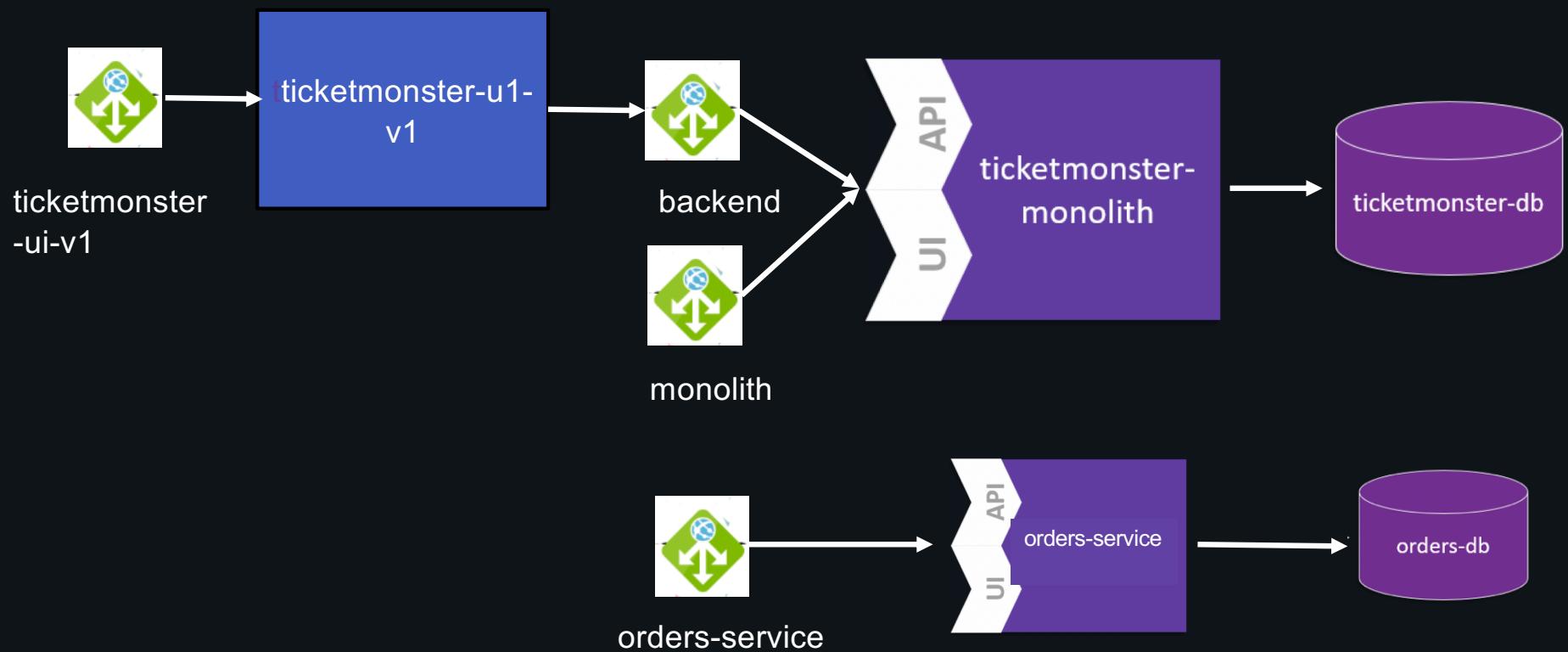
Create route for the orders microservice

- 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:

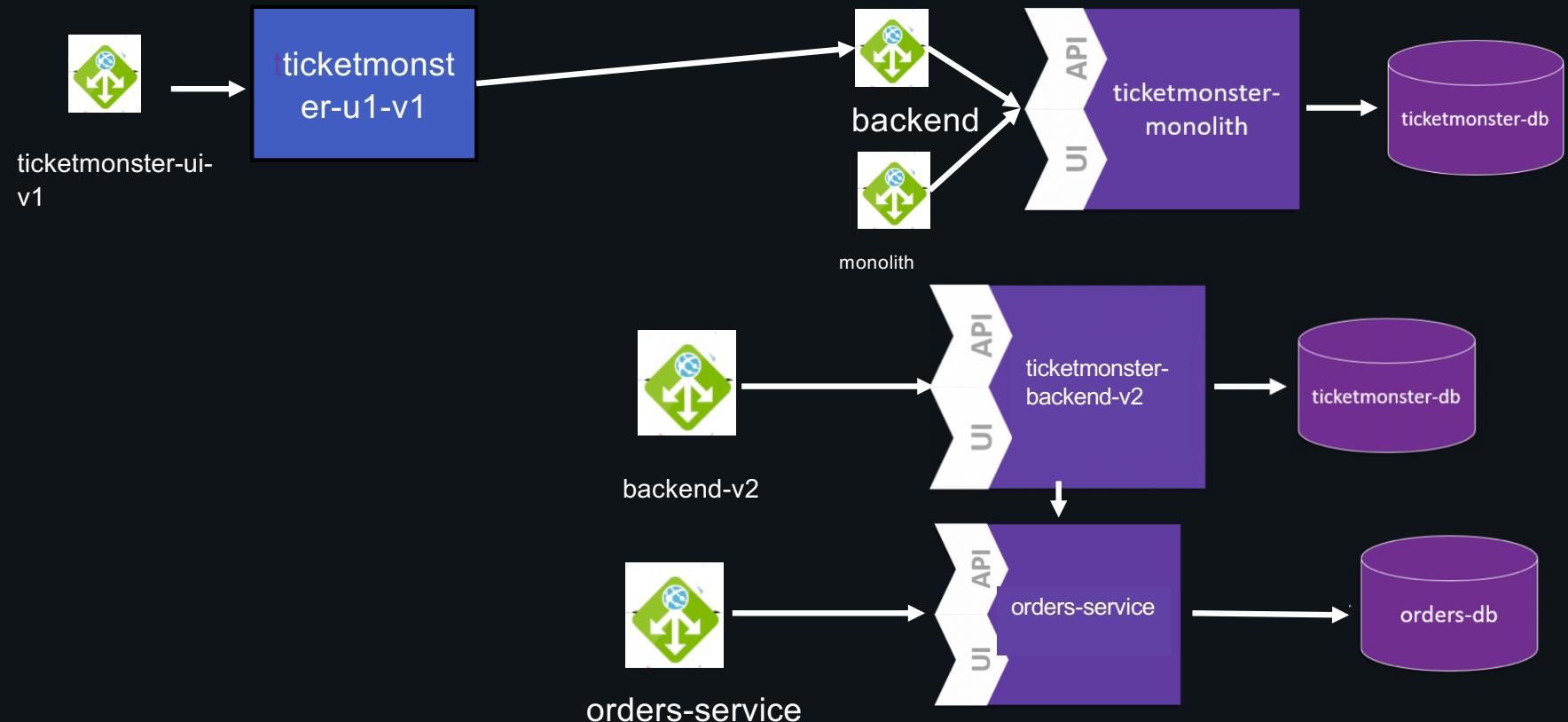
Create route for ticketmonster-backend-V2

- 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:

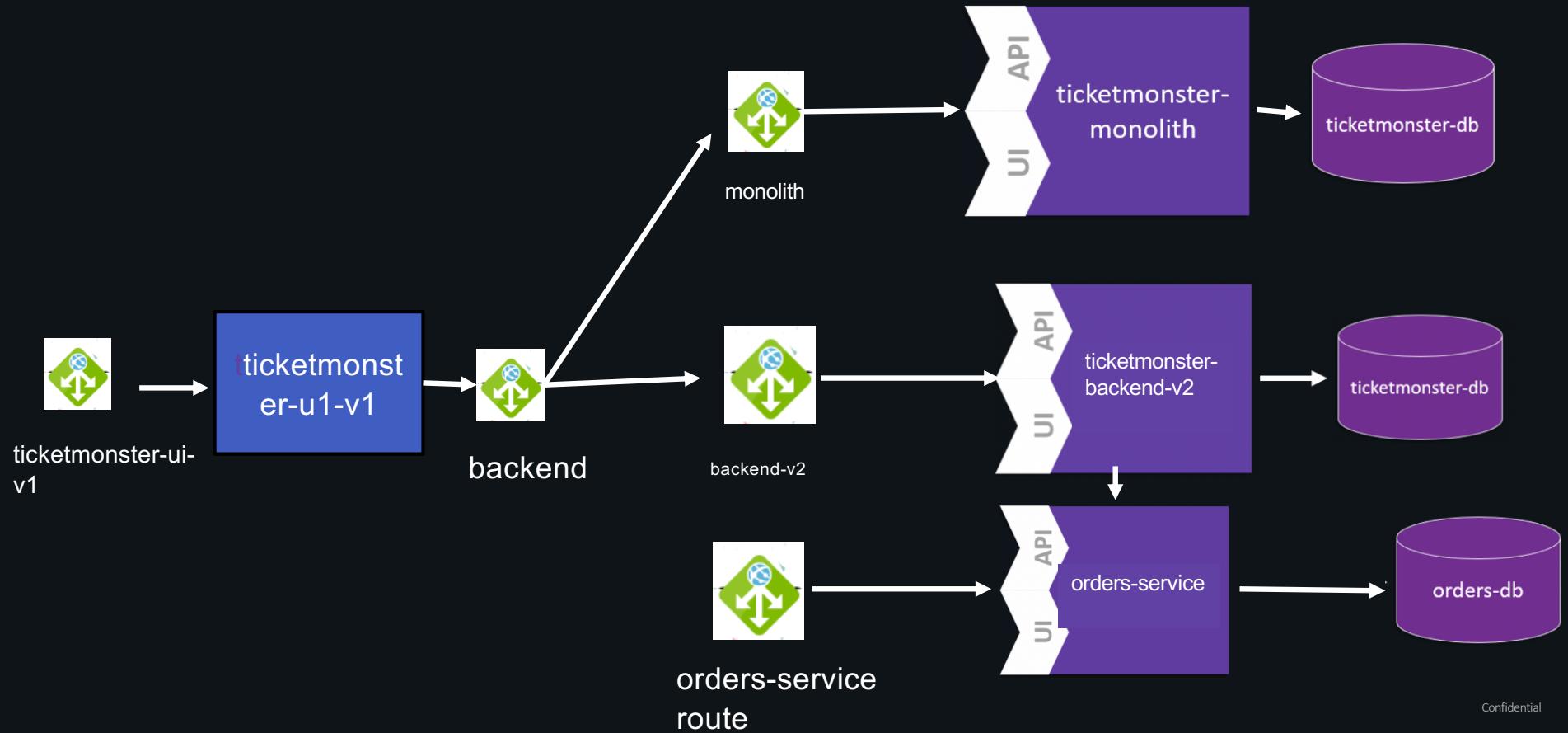
Split route to backend

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.

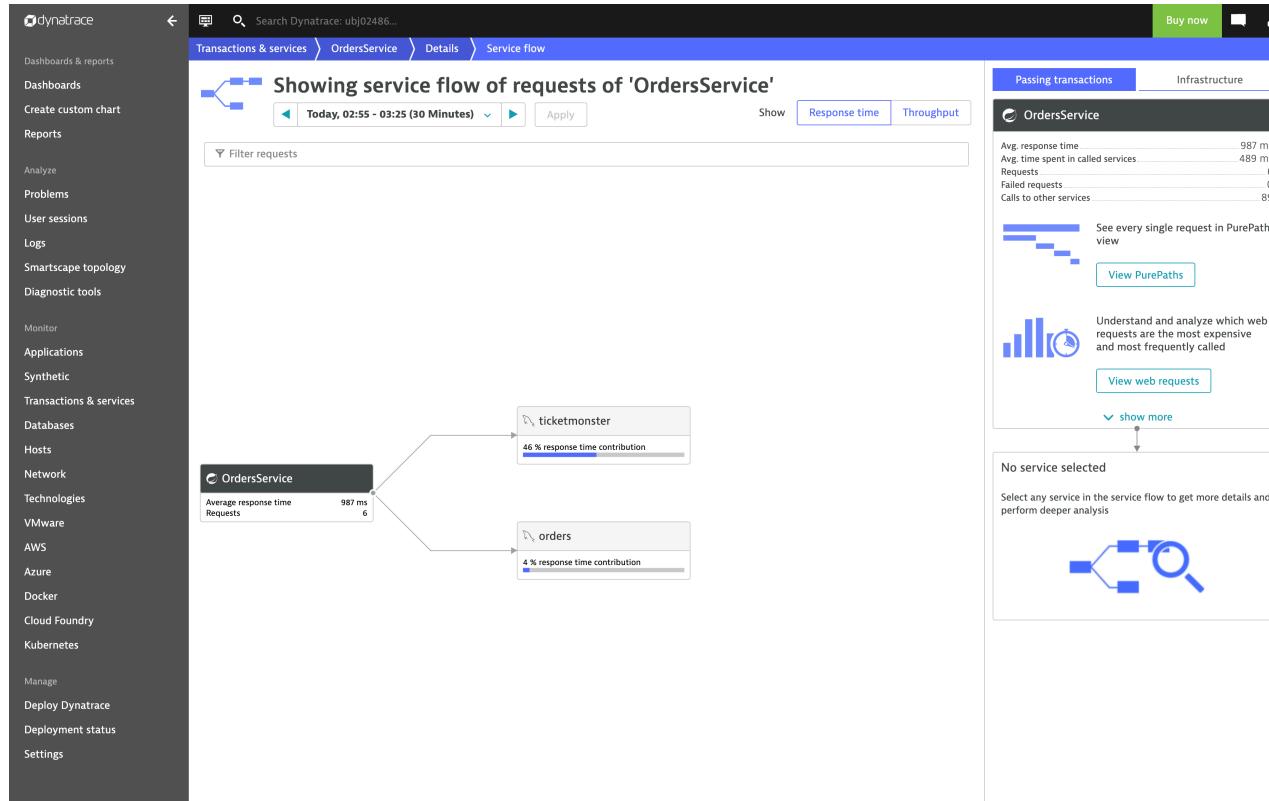
Finally, you see the service flow containing the microservice orders-service.

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](https://github.com/Dynatrace-Asad-Ali/Monolith_to_Microservices/tree/master/6%20Deploy%20the%20Microservice)