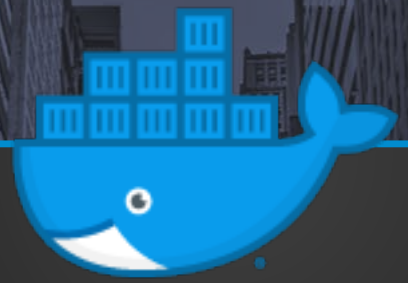
DOCKER AND KUBERNETES

VITALS CHECK

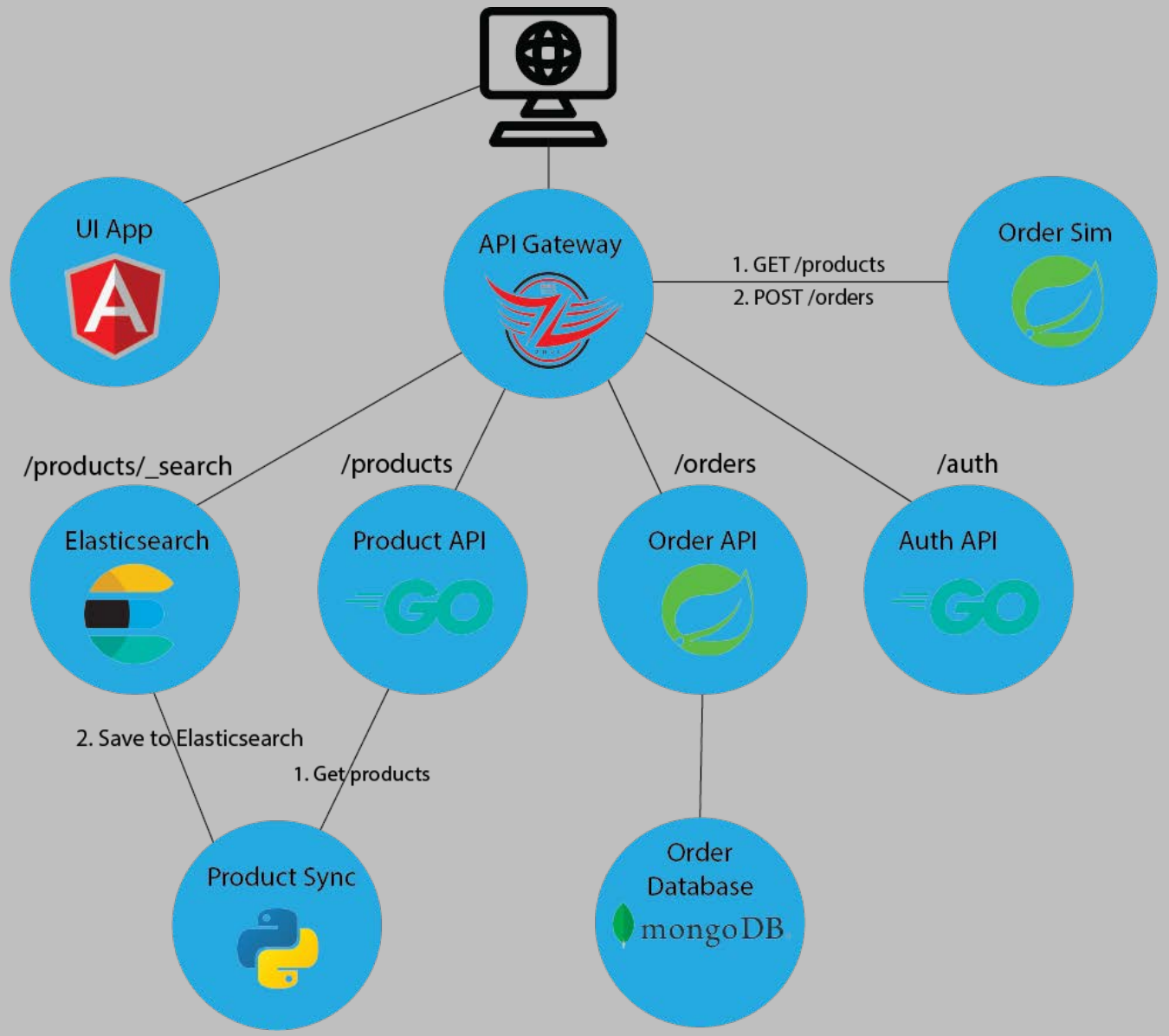
- How are we feeling about the class so far?
- Lost on anything?
- Questions from yesterday?
- Are you happy with the teaching style so far?
- What adjustments can I make to help?
- Is the pace OK?

RV STORE KUBERNETES HACKATHON

RV STORE



RVSTORE - OVERVIEW



HACKATHON - OVERVIEW

- The RV store is a mock ecommerce application.
- Your task is to get the application running on a Kubernetes cluster.
- The application has the following services, each with their own Docker image:
  - Angular UI running in Nginx
  - Authentication service
  - Product service
  - Order service
  - Order simulator
  - Gateway edge service
  - Product sync service
  - Product search service (Elasticsearch)
- Solutions are provided in the Github repo. But try to only use them to get unstuck on a specific problem!
- Github repo is at https://www.github.com/VergeOps/k8s-rvstore

HACKATHON - OBJECTIVES

Your humble instructor is playing the role of developer. I've written an application made up of 6 services. But I need your expertise to get it running on Kubernetes. All I know is the application code and environment variables needed.

Your goals are (in order of importance):

1. Set up the application to run in Kubernetes. For this hackathon, Minikube or Docker Kubernetes for Desktop is fine.
2. Use ConfigMaps to provide environment variables, inject product data, and put any sensitive information into secrets.
3. Ensure that only public services are accessible outside the cluster. These are the gateway service and the UI.
4. Make the app fault-tolerant
    1. Multiple copies of services
    2. Set up probes
5. For MongoDB, set up a volume mapping to your hard drive so that the MongoDB pod can be thrown out and not lose orders.
6. If we covered HorizontalPodAutoscaler in this class, try adding scaling to one of your deployments, like the product API.

HACKATHON – LEARNING THROUGH THE PAIN

Exercises so far have been very simple and superficial. This is by design, as I want you to get the deep dive knowledge from this hackathon.

This hackathon is designed to push you. It is intended to make you a little uncomfortable. You may not enjoy it (at least until the end when you have it working)!

The struggle is where the learning is. You will scratch your head, wonder what's going on. This is designed to mimic real life so that you can troubleshoot, then come to me (the developer) to get the proper information.

Past classes have overwhelmingly told me that this is the best part of the class because students come away with a solid foundation of Docker and Kubernetes and have confidence that they can go implement a real application.

HACKATHON – HELPFUL HINTS

It is best to start out as simple as possible. Eliminate any variables that might muddy up what you're doing.

Pick a service that is the simplest and start there. Implement it, get it running, then move on. Don't try to just write all the files at once then wonder why things aren't working. Build from simple to complex in an iterative process. The UI service is a good place to start since it just serves static information and has no dependencies on other services.

Save things like fault-tolerance for later. Don't use multiple copies of a service yet. Don't add probes. Save that for once it's working.

Don't forget that you can test services directly by making them NodePort, hitting them from other pods, or using kubectl port-forward.

RV STORE – UI APPLICATION

- This is an Angular application running nginx to serve the files
- The application serves at port 80
- This application should be publicly accessible
- Docker image: public.ecr.aws/e7e6w2e3/rvstore-ui
- No environment variables needed
- The UI gets its data by making HTTP calls to the backend gateway API.
  - <backend>/products to get product information
  - <backend>/products/_search to search the Elasticsearch instance
  - <backend>/orders to get order information
  - <backend>/auth to get auth information
- In the UI itself, there is a text box to enter the base URL of the backend gateway service. Note that it must include the trailing slash.

RV STORE – PRODUCT API APPLICATION

- This is a Golang application. It serves up the product information as a REST API.
- The application serves at port 9001
- The application should only be accessible inside the cluster
- Docker image: public.ecr.aws/e7e6w2e3/rvstore-product-api
- You must provide an environment variable specifying the internal directory location of the product data: PRODUCT_FILE_LOCATION. I suggest /data/products
- You must provide the products.json file to the container in a ConfigMap. Place it inside the container at the same location as the PRODUCT_FILE_LOCATION you gave above.
  - The products.json file can be found in the exercise files in the rvstore_hackathon directory.
- You can test the service at http://<service>/products

RV STORE – AUTHENTICATION API APPLICATION

- This is a Golang application. It serves up a JSON Web Token (JWT) in response to a login attempt. It does not take a username/password, but instead gives back a JWT any time the login endpoint is called.
- The application serves at port 9003
- The application should only be accessible inside the cluster
- Docker image: public.ecr.aws/e7e6w2e3/rvstore-auth-api
- You can test the service at http://<service>/auth/login

RV STORE – ORDER API APPLICATION

- This is a Java Spring Boot application. It receives order data and stores it in the Mongo database
- The application serves at port 9002
- The application should only be accessible inside the cluster
- It communicates with the Mongo service by name rvstore-orders-mongodb
- Docker image: public.ecr.aws/e7e6w2e3/rvstore-order-api
- Environment variables needed:
  - SPRING_PROFILES_ACTIVE: compose
- You can test the service at http://<service>/orders

RV STORE – ORDER SIMULATOR APPLICATION

- This is a Java Spring Boot application. It generates random orders and submits them to the order API.
- There is no port number for this app. It is not a web app but instead just a background process.
- It communicates with the Gateway service at: http://rvstore-api-gateway:9000
- This pod should run as a Kubernetes BatchJob, running about once a minute.
- Docker image: public.ecr.aws/e7e6w2e3/rvstore-order-simulator
- Environment variables needed:
  - SPRING_PROFILES_ACTIVE: compose
  - JOB: true

RV STORE – ELASTICSEARCH

- This is the stock Elasticsearch image from Docker Hub. It stores products to make them searchable. The product sync service populates it with products from the product service.
- The application listens on port 9200
- Run it as a StatefulSet with a single replica.
- It communicates with the product service at: http://rvstore-product-api:9001 and the Elasticsearch service at http://elasticsearch:9200.
- Docker image: elasticsearch:7.12.0
- Environment variables:
  - discovery.type=single-node
  - ES_JAVA_OPTS=-Xms256m -Xmx256m
  - http.cors.allow-origin="*"
  - http.cors.enabled="true"
  - http.cors.allow-headers=X-Requested-With,X-Auth-Token,Content-Type,Content-Length,Authorization
  - http.cors.allow-credentials="true"

RV STORE – API GATEWAY APPLICATION

- This is a Java Spring Boot application. It routes traffic to the appropriate application based on the path. It acts as traffic cop. For example, xyz.com/products will get routed to the product API application
- Runs on port 9000
- Application should be publicly accessible as the only endpoint for the backend API
- It communicates with other services:
  - Auth service at: http://rvstore-auth-api:9003/auth
  - Product service at: http://rvstore-product-api:9001/products
  - Order service at: http://rvstore-order-api:9002/orders
- Docker image: public.ecr.aws/e7e6w2e3/rvstore-gateway-service
- Environment variables needed:
  - SPRING_PROFILES_ACTIVE: compose

RV STORE – PRODUCT SYNC APPLICATION

- This is a Python application. It reads the products from the product service and pushes them to Elasticsearch
- The application does not listen on a port
- The application can be run on a schedule using a BatchJob.
- It communicates with the product service at: http://rvstore-product-api:9001 and the Elasticsearch service at http://elasticsearch:9200.
- Docker image: public.ecr.aws/e7e6w2e3/rvstore-product-sync
- Environment variable:
  - JOB: “true”

RV STORE – MONGODB DATABASE

- For this we're using the public mongo image in Docker Hub.
- Docker image: public.ecr.aws/e7e6w2e3/rvstore-mongo
- Runs on port 27017
- Run it as a StatefulSet with a single replica.
- Should be accessible only within the cluster
- Mongo stores data internally at /data/db
- Environment variables needed:
  - MONGO_INITDB_ROOT_USERNAME: mongoadmin
  - MONGO_INITDB_ROOT_PASSWORD: secret