END USER LICENSE AGREEMENT

Enacted January 1, 2019

IMPORTANT: This agreement is the end user license agreement (hereinafter referred to as this “Agreement”) between you (an individual) and VergeOps, LLC (hereinafter referred to as “VergeOps”) for use of this training content and any related materials, source code, diagrams, and documentation.

This end user license agreement is between you and VergeOps and governs your use of this training content.

License to use the content: VergeOps grants you a nonexclusive, nontransferable, revocable license to use the training content and related materials, source code, diagrams, and documentation as permitted by this Agreement solely for personal and noncommercial use. This training content cannot be used for any other purpose.

Limitations of use: You are not permitted to lease, rent, sublicense, publish, copy, modify, adapt, translate, reverse engineer, decompile, or disassemble all or a portion of this content without VergeOps’ prior written consent or unless otherwise expressly permitted by applicable law.

Termination and ongoing effectiveness: This Agreement is effective from the first date you receive this training content. You may terminate this Agreement at any time by permanently deleting and destroying this training content and any related materials, source code, diagrams, and documentation. VergeOps may terminate this Agreement at any time without notice if you fail to comply with any terms or conditions of this Agreement. Once it is terminated, you MUST stop using this training content and any related materials, source code, diagrams, and documentation and delete all training content and any related materials, source code, diagrams, and documentation immediately.

Disclaimer of warranties: You acknowledge that the training content and any related materials, source code, diagrams, and documentation is provided “as is” without warranty of any kind, express or implied, and to the maximum extent permitted by applicable law. Neither VergeOps, its licensors or affiliates nor the copyright holders make any representations or warranties, express or implied.

Limitation of liability: To the fullest extent of applicable law, licensor shall not be liable for special, incidental, or consequential damages resulting from possession, use, or malfunction of the software, including but not limited to damages of property, loss of goodwill, computer failure or malfunction. To the fullest extent of applicable law, licensor’s liability for all damages shall not (except as required by applicable law), exceed the actual price paid by you (if any) for use of the training content and related materials, source code, diagrams, and documentation.

Governing Law: This Agreement is entered into in the State of Colorado and shall be governed by, and construed with, the laws of the State of Colorado, exclusive of its choice of law rules.

Any questions about these terms or our services can be directed to VergeOps by email admin@vergeops.com.

VergeOps

VergeOps

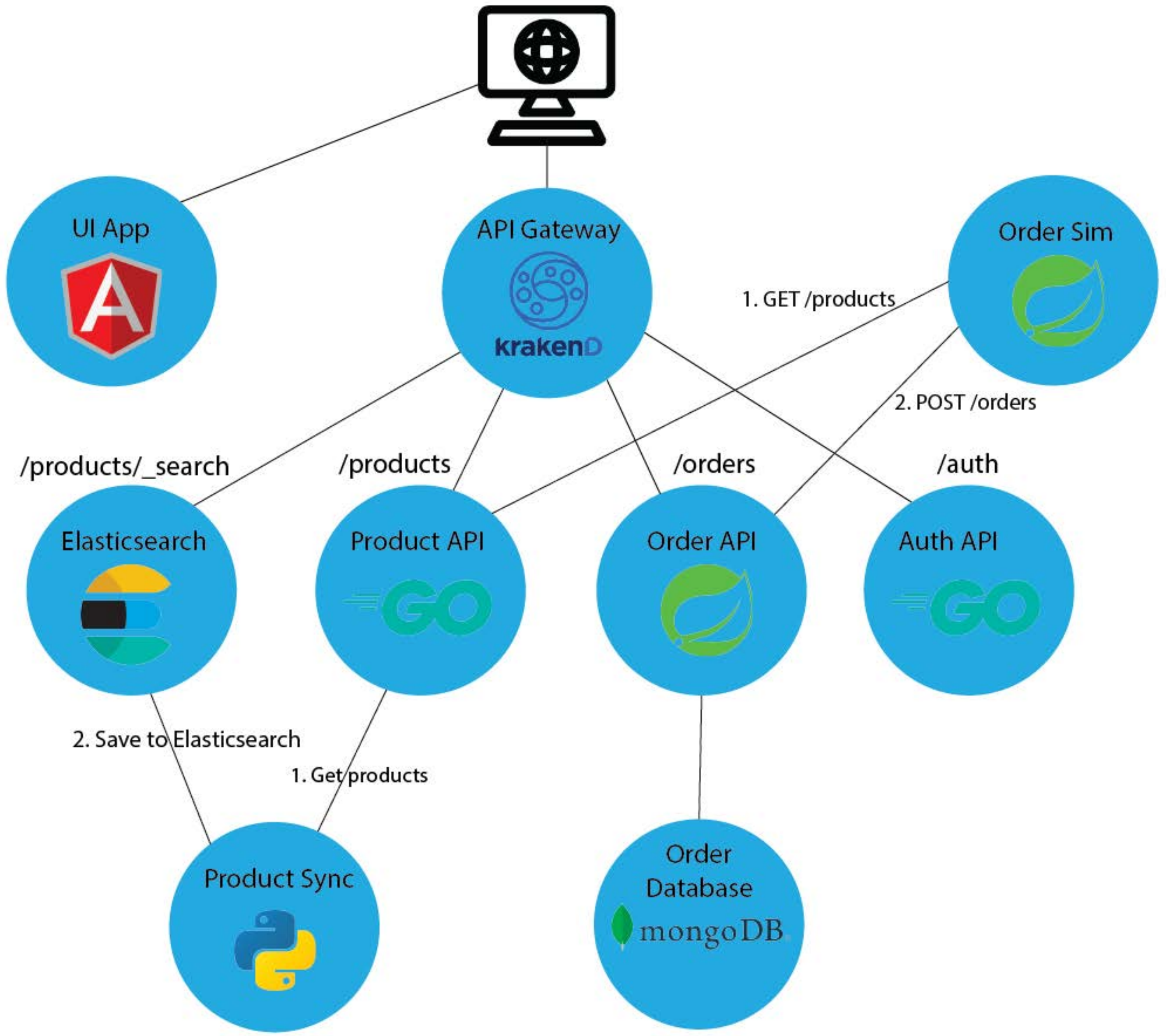
RV Store Hackathon

with Tim Solley

VergeOps

RV Store Overview

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

OBJECTIVES

Your humble instructor is playing the role of developer. I've written an application made up of 8 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. Just get it working completely.
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 and resilient to failure. Try to break it!
    1. Multiple replicas of pods
    2. Set up probes
5. (Optional) 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. (Optional) If we covered HorizontalPodAutoscaler in this class, try adding scaling to one of your deployments, like the product API.

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.

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.

NOTE: In the rvstore_hackathon directory is another directory named working_dir. I suggest you create your yaml files there. You'll find a couple files already there to get you started.

VergeOps

Microservices

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/vergeops/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.

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/vergeops/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

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/vergeops/rvstore-auth-api
- You can test the service at http://<service>/auth/login

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/vergeops/rvstore-order-api
- Environment variables needed:
  - SPRING_PROFILES_ACTIVE: compose
- You can test the service at http://<service>/orders

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 Product service at: http://rvstore-product-api:9001 and the order service at http://rvstore-order-api:9002
- This pod should run as a Kubernetes CronJob, running about once a minute.
- Docker image: public.ecr.aws/vergeops/rvstore-order-simulator
- Environment variables needed:
  - SPRING_PROFILES_ACTIVE: compose
  - JOB: "true"

API GATEWAY APPLICATION

- This applications uses the open source project KrakenD from the Cloud Native Compute Foundation (the same foundation that owns Kubernetes). 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
  - Elasticsearch at: http://elasticsearch:9200
- Docker image: public.ecr.aws/vergeops/rvstore-gateway-service

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 CronJob.
- 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/vergeops/rvstore-product-sync
- Environment variable:
    - JOB: "true"

ELASTICSEARCH

- NOTE: There is already a file named elasticsearch.yaml in the working directory. You can just run it to have a working 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.
- Docker image: elasticsearch:7.13.4
- 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"

MONGODB

- NOTE: There is already a file named mongo-stack.yaml in the working directory. You can just run it to have a working MongoDB.
- For this we're using the public mongo image in Docker Hub.
- Docker image: public.ecr.aws/vergeops/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