Member-only story

Practical Kubernetes. Deploy User Management Microservice.

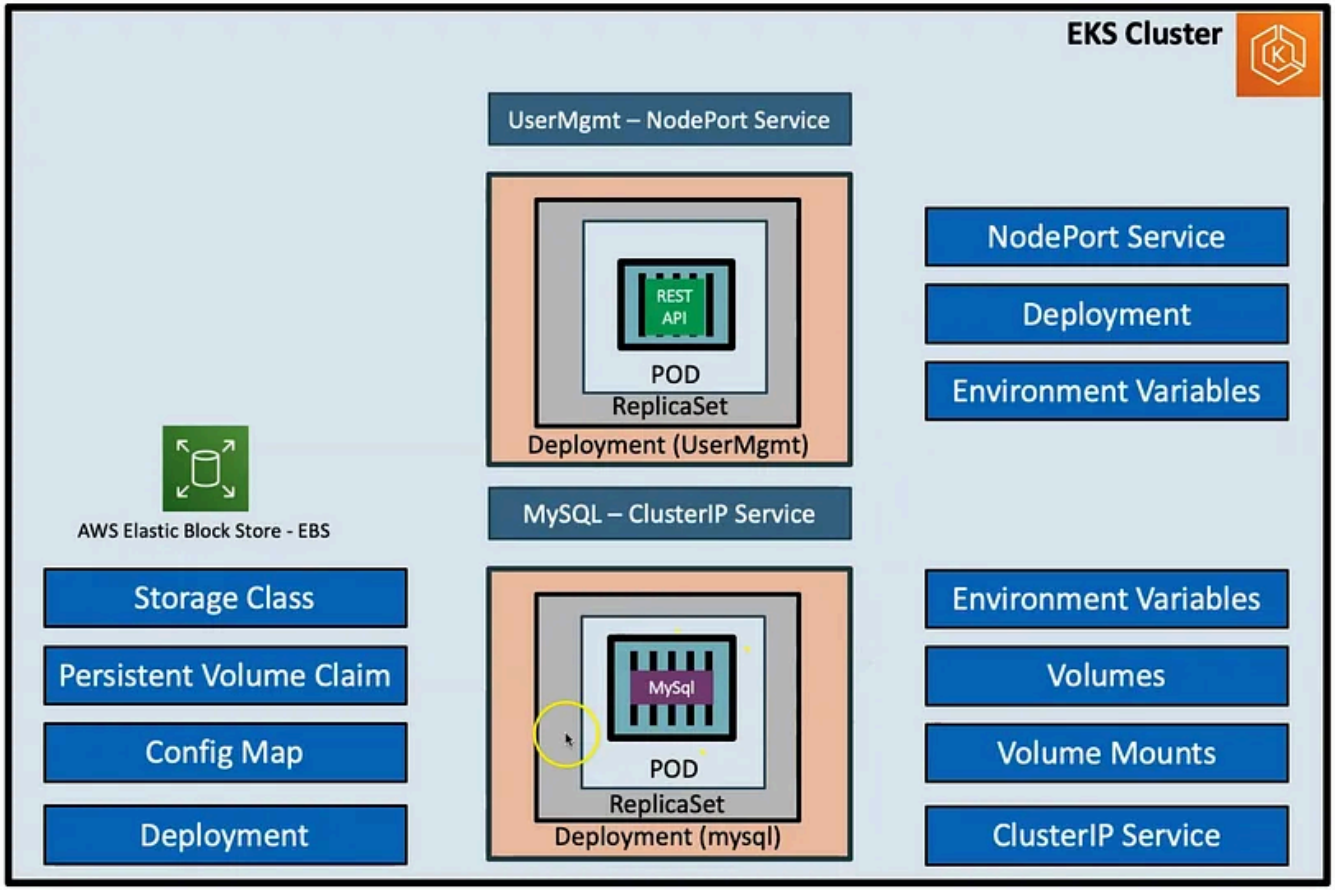
Konstantin Mogilevskii · Follow

Published in Dev Genius · 8 min read · 4 days ago

kubernetes

Introduction

The goal of this article is to learn Kubernetes by doing. Particularly, we're going to deploy the User Management REST API that utilizes MySQL DB. Here's the architecture overview



Namespace, LimitRange and ResourceQuota

- Namespaces allow to split-up resources into different groups.
- Resource names should be unique in a namespace.
- We can use namespaces to create multiple environments like dev, staging and production etc.

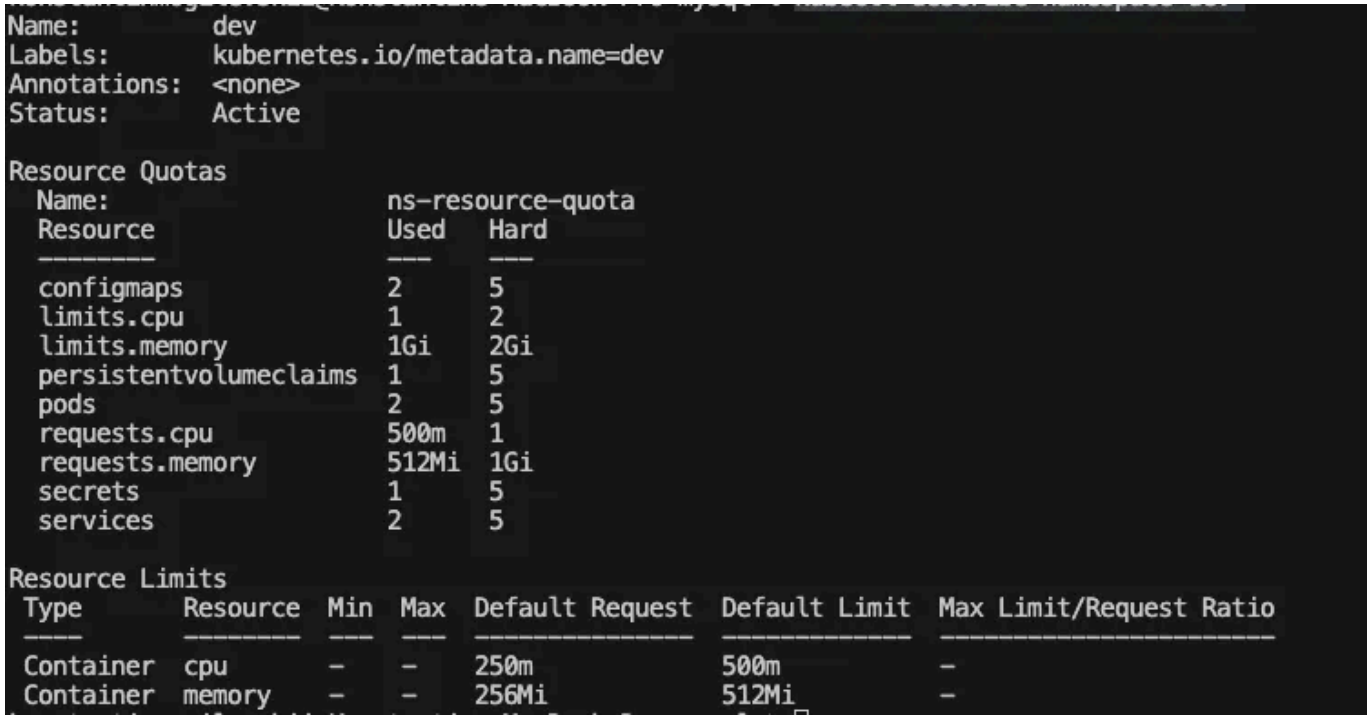
- Kubernetes will always list the resources from `default namespace` unless we provide exclusively from which namespace we need information from.
- Instead of specifying `resources like cpu and memory` in every container spec of a pod definition, we can provide the default CPU & Memory for all containers in a namespace using `LimitRange`.
- ResourceQuota allows us to specify the limits (in terms of CPU, Memory, and number of default kubernetes resources) on the namespace as a whole.

```
apiVersion: v1
kind: Namespace
metadata:
  name: dev
---
apiVersion: v1
kind: LimitRange
metadata:
  name: default-cpu-mem-limit-range
  namespace: dev
spec:
  limits:
    # every pod (container) in dev namespace won't consume more than this
    - default:
        memory: 512Mi
        cpu: 500m
```

```
      # every pod (container) in dev namespace allocate at least this amount mem
      defaultRequest:
        memory: 256Mi
        cpu: 250m
      type: Container
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: ns-resource-quota
  namespace: dev
spec:
  # namespace will take minimum of 1 vCPU and 1Gi of RAM
  # and won't consume more than 2 vCPUs and 2Gi of RAM.
  # We also specify the limits on number of particular k8s resources
  # that can be deployed in dev namespace
  hard:
    limits.cpu: 2
    limits.memory: 2Gi
    requests.cpu: 1
    requests.memory: 1Gi
    pods: 5
    configmaps: 5
    services: 5
    secrets: 5
    persistentvolumeclaims: 5
```

Verify the Resource Quotas and Limit Ranges

```
kubectl describe namespace dev
```



```
Name:         dev
Labels:       kubernetes.io/metadata.name=dev
Annotations:  <none>
Status:       Active

Resource Quotas
  Name:                   ns-resource-quota
  Resource                Used   Hard
  --------                ---    ---
  configmaps              2      5
  limits.cpu              1      2
  limits.memory           1Gi    2Gi
  persistentvolumeclaims  1      5
  pods                    2      5
  requests.cpu            500m   1
  requests.memory         512Mi  1Gi
  secrets                 1      5
  services                2      5

Resource Limits
 Type       Resource  Min  Max  Default Request  Default Limit  Max Limit/Request Ratio
 ----       --------  ---  ---  ---------------  -------------  -----------------------
 Container  cpu       -    -    250m             500m           -
 Container  memory    -    -    256Mi            512Mi          -
```

StorageClass

- Instead of creating **PersistentVolume**, that MySQL deployment will request with **PersistentVolumeClaim**, we'll allow Kubernetes to dynamically allocate persistent storage from attached EBS volumes to our EC2 instances that represent Worker Nodes of Kubernetes cluster.

- This is done via StorageClass resource that doesn't belong to any namespace.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com # AWS EBS provision our persistent storage
volumeBindingMode: WaitForFirstConsumer # k8s won't allocate any volume unless t
```

PersistentVolumeClaim

- StorageClass just allows us to utilize the underlying volume that is binded to our EC2 instance. In order for a pod to use storage, it needs to make a request for it specifying the desired amount

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-mysql-pv-claim
  namespace: dev
spec:
  accessModes:
```

```
    - ReadWriteOnce
  storageClassName: ebs-sc # refer to the StorageClass we created previously
  resources:
    requests:
      storage: 4Gi # specify the amount of volume we need
```

ConfigMap

- Kubernetes resource for storing non-sensitive configuration data for our deployed microservices.
- When we pass values to our environmental variables for pods, we refer to the keys mentioned in **ConfigMap**.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: usermanagement-dbcreation-script
  namespace: dev
data:
  # this is gonna be a SQL-script that will configure our MySQL DB on the start.
  mysql_usermgmt.sql: |-
    DROP DATABASE IF EXISTS usermgmt;
    CREATE DATABASE usermgmt;
```

MySQL Deployment

- Most of the time we don't deploy pods directly.
- We utilize the **Deployment** k8s resource for managing maintaining pods in a desired state.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  namespace: dev
spec:
  # maintain just 1 replica of a pod
  replicas: 1
  strategy:
    # if we upgrade the pod, the old one will be first dropped
    # and afterwards a new one will spin up
    type: Recreate
  selector:
    # deployment will be responsible for the pods
    # with the labels app-mysql
    matchLabels:
      app: mysql
  template:
    metadata:
      # lables attached to the actual pod
      # should correspond to the one deployment specifies
      labels:
        app: mysql
    spec:
      containers:
```

```
      - name: mysql
        image: mysql:5.6
        ports:
          - containerPort: 3306
            name: mysql
        # actual place within the file system of a pod
        # where we bind our volumes
        volumeMounts:
          - name: mysql-persistent-storage
            mountPath: /var/lib/mysql
          - name: usermgmt-dbcreation-script
            mountPath: /docker-entrypoint-initdb.d
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              # value of password is a sensitive info
              # so it can't be stored in ConfigMap
              # we utilize another k8s resource - Secret
              secretKeyRef:
                name: mysql-secrets
                key: mysql-db-password
    # we list volumes that our pod will require
    volumes:
        # first is a request for the actual storage of a DB data
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: ebs-mysql-pv-claim
        # second is the SQL-script we need to execute on a startup
        # that's stored in ConfigMap
      - name: usermgmt-dbcreation-script
        configMap:
          name: usermanagement-dbcreation-script
```

Verify MySQL Deployment

```
# connect to our mysql pod
kubectl exec -it -n dev mysql-698ff4d9f8-dkmdr bash

# connect to the mysql server running within it
mysql -u root -pdbpassword11

# you should see our database we specified in configmap
mysql> show schemas;
```

ClusterIP Service for MySQL Deployment

- Pods are naturally exposable and there're tons of reasons to restart them.
- Once the pod is restarted it changes its internal-IP. Hence other pods that were utilizing this IP directly will fail to communicate.
- We need something on top of a Pod for communication that won't change on Pod restarts.
- ClusterIP Service is exactly for that. We won't be able to access our Pod from outside of a k8s cluster, but internal communication between pods will happen through this service.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql
  namespace: dev
spec:
  selector:
    app: mysql
  ports:
    - port: 3306
  clusterIP: None
```

Verify the MySQL Deployment via service

```
# we specify host as the name of ClusterIP service
kubectl run -it --rm --image=mysql:5.6 --restart=Never \
              -n dev  mysql-client01 -- mysql -h mysql -pdbpassword11
```

Secret

- Just like a ConfigMaps, but for storing a sensitive information within a Kubernetes cluster.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secrets
  namespace: dev
type: Opaque
data:
  # we specify key-value pairs of sensitive info
  # where values are not stored in a raw format, but base64-encoded
  mysql-db-user: cm9vdA== # base64 encoded root
  mysql-db-password: ZGJwYXNzd29yZDEx
  mysql-db-hostname: bXlzcWw= # base64 encoded mysql value
  mysql-db-host: bXlzcWw=
  mysql-db-name: dXNlcm1nbXQ= # base64 encoded usermgmt value
```

User Management Deployment

- This is our REST API that will communicate with MySQL Deployment for persistently storing information about users.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: usermgmt
  namespace: dev
spec:
  replicas: 1
```

```
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: usermgmt
  template:
    metadata:
      labels:
        app: usermgmt
    spec:
      # specify container that spin up first before our main app process
      # it'll run in a loop checking the availability of a MySQL deployment
      # once MySQL respond back, we stop the init container and start the main o
      # this used to prevent unnecessary pod restarts, when depended pod isn't u
      initContainers:
        - name: init-db
          image: busybox:1.31
          command: ['sh', '-c', 'echo -e "Checking for the availability of MySQL
      containers:
        - name: usermgmt
          image: stacksimplify/kube-usermanagement-microservice:1.0.0
          # bunch of env vars that aren't directly specified here
          # but actually stored in a secret and referenced here
          env:
            - name: DB_HOSTNAME
              valueFrom:
                secretKeyRef:
                  name: mysql-secrets
                  key: mysql-db-hostname

            - name: DB_PORT
              value: "3306"

            - name: DB_NAME
              valueFrom:
                secretKeyRef:
```

```
            name: mysql-secrets
            key: mysql-db-name

      - name: DB_USERNAME
        valueFrom:
          secretKeyRef:
            name: mysql-secrets
            key: mysql-db-user

      - name: DB_PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysql-secrets
            key: mysql-db-password

    ports:
      - containerPort: 8095
    # once 60 seconds after the start of a container passes we run
    # a check every 10 seconds to make sure the process is up and running
    # without being stuck in any kind of a deadlock
    livenessProbe:
      exec:
        command:
          - /bin/sh
          - -c
          - nc -z localhost 8095
      initialDelaySeconds: 60
      periodSeconds: 10
    # this check that process is not only up and running
    # but also accepts the HTTP traffic
    readinessProbe:
      httpGet:
        path: /usermgmt/health-status
        port: 8095
      initialDelaySeconds: 60
      periodSeconds: 10
```

NodePort Service for User Management

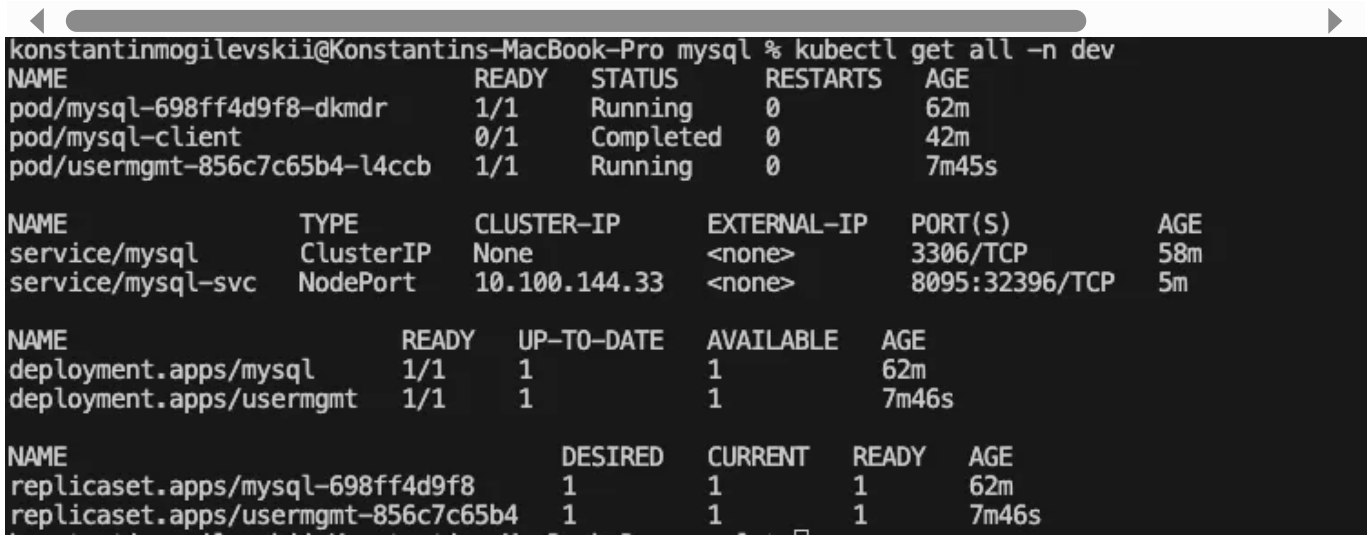
- We want t be able to access our User Management Deployment from outside of a k8s cluster.
- ClusterIP Service isn't suitable for it.
- NodePort is a simplest way to expose our Deployment to outside world. It basically opens a specific port on our Worker Nodes and all traffic that comes to it will be redirected to this service.
- Don't forget to add new inbound rules for Security Groups attached to EC2 instances that represent our Node Group. They needs to allow any traffic to the port you specify in a **nodePort** key.

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-svc
  namespace: dev
  labels:
    app: usermgmt
spec:
```

```
  type: NodePort
  ports:
    - port: 8095
      targetPort: 8095
      nodePort: 31231 # we can drop it and k8s will allocate port randomly in a
  # specify to which pods traffic will be forwarded based on the labels of a pod
  selector:
    app: usermgmt
```

Verify the whole deployment

Run the **kubectl get all -n dev** to list all the deployed resources in our dev namespace

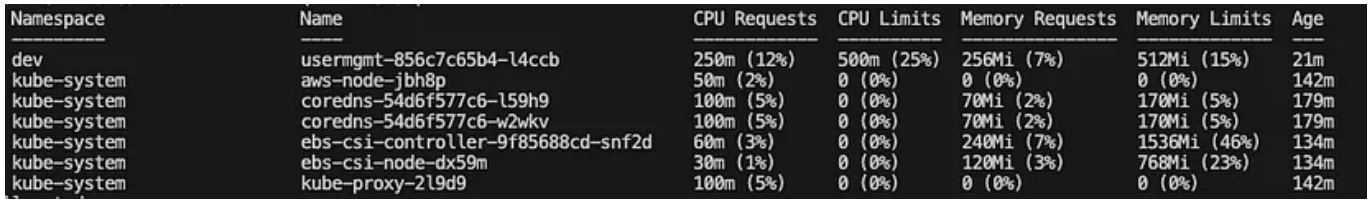


As you can see **mysql-svc** Service forwards traffic from the port 32396 to 8095 of a Pod that runs User Management system. That's because in the actual steps I commented the nodePort and allowed k8s to pick one randomly for me.

Let's check how many EC2 instance resources are consumed by our deployments:

```
kubectl get nodes

kubectl describe node <node-name>
```



Namespace	Name	CPU Requests	CPU Limits	Memory Requests	Memory Limits	Age
dev	usermgmt-856c7c65b4-l4ccb	250m (12%)	500m (25%)	256Mi (7%)	512Mi (15%)	21m
kube-system	aws-node-jbh8p	50m (2%)	0 (0%)	0 (0%)	0 (0%)	142m
kube-system	coredns-54d6f577c6-l59h9	100m (5%)	0 (0%)	70Mi (2%)	170Mi (5%)	179m
kube-system	coredns-54d6f577c6-w2wkv	100m (5%)	0 (0%)	70Mi (2%)	170Mi (5%)	179m
kube-system	ebs-csi-controller-9f85688cd-snf2d	60m (3%)	0 (0%)	240Mi (7%)	1536Mi (46%)	134m
kube-system	ebs-csi-node-dx59m	30m (1%)	0 (0%)	120Mi (3%)	768Mi (23%)	134m
kube-system	kube-proxy-2l9d9	100m (5%)	0 (0%)	0 (0%)	0 (0%)	142m

One node will have MySQL deployment and other the UserManagement Deployment. No matter what the process is doing it still consumes at minimum the 250m vCPU and 256Mi of RAM.

Once you altered the Security Group Inbound Rules for our Node Group, we can check the API with several calls:

```
GET request to http://34.229.193.196:32396/usermgmt/health-status

POST request to http://34.229.193.196:32396/usermgmt/user to create a user and s
Body JSON can be
    {
        "username": "admin1",
        "email": "dkalyanreddy@gmail.com",
        "role": "ROLE_ADMIN",
        "enabled": true,
        "firstname": "fname1",
        "lastname": "lname1",
        "password": "Pass@123"
    }

GET to http://34.229.193.196:32396/usermgmt/users to list all the present users
```

Thanks!

Kubernetes Microservices Aws Eks

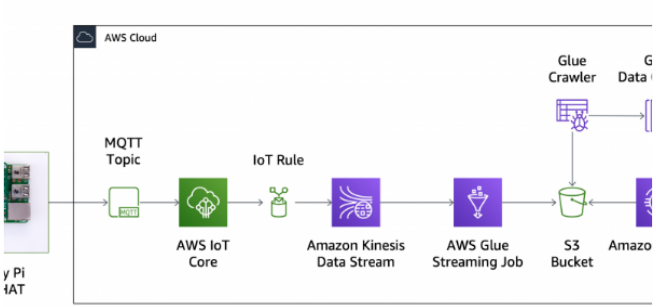
Written by Konstantin Mogilevskii

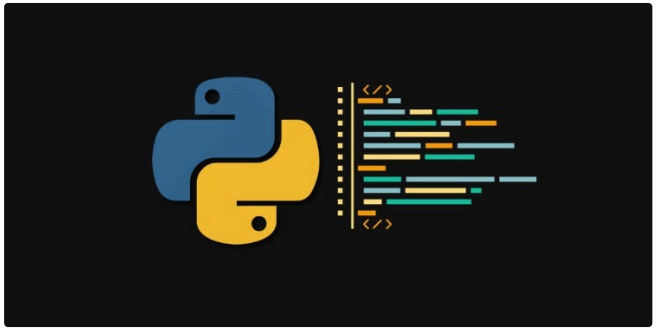
Follow

230 Followers · Writer for Dev Genius

Senior Data Engineer www.linkedin.com/in/kmogilevskii

More from Konstantin Mogilevskii and Dev Genius





Konstantin Mogilevskii in Dev Genius

Serverless Streaming ETL with AWS Glue

Introduction

7 min read · May 9, 2024

27

Tomas Svojanovsky in Dev Genius

Pytest Fixtures: Your Secret Weapon for Writing Powerful Tests

Exploring Pytest Fixtures: Setup, Teardown, Scopes, and Best Practices

6 min read · Apr 15, 2024

385 4

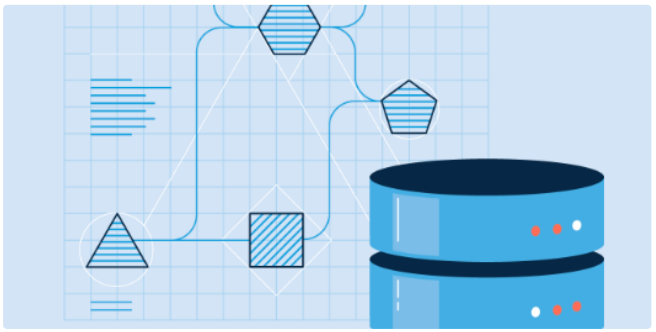
Ani Talakhadze in Dev Genius

Top 10 Essential Stream Operations in Java

Boost Your Java Efficiency With Examples, Exercises And Fun Facts

9 min read · Apr 21, 2024

80 4



Konstantin Mogilevskii in Dev Genius

Practical Data Modeling with Data Vault 2.0 and Vertica

Introduction

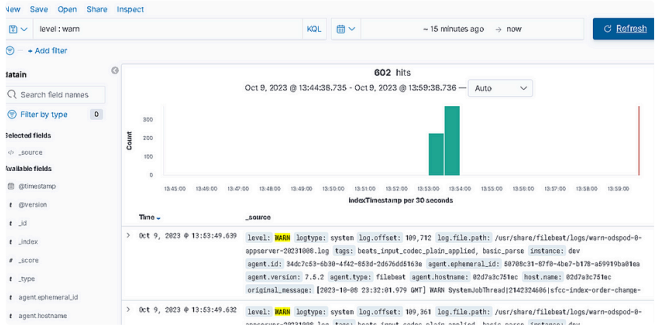
6 min read · Mar 23, 2024

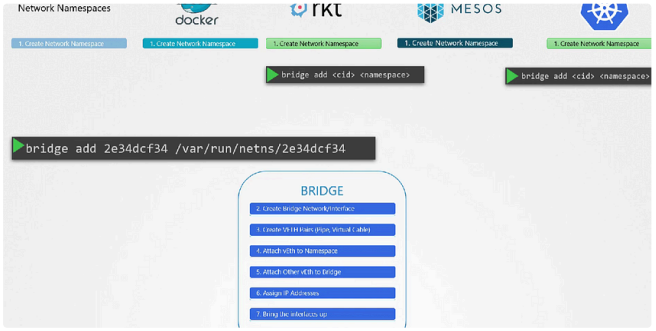
51

See all from Konstantin Mogilevskii

See all from Dev Genius

Recommended from Medium





Yassine Elmahi

Using a Dockerized ELK-F stack to consume and analyse logs

What is the ELK stack?

8 min read · 3 days ago

Devendra Johari in DevOps.dev

Networking in Kubernetes

Kubernetes

18 min read · May 1, 2024

91

Lists

Natural Language Processing

1456 stories · 966 saves

Tanat Lokejaroenlarb in NonTechCompany

ArgoCd ApplicationSet is more practical in version v2.9

3 min read · Apr 27, 2024

25

Tamer Benhassan

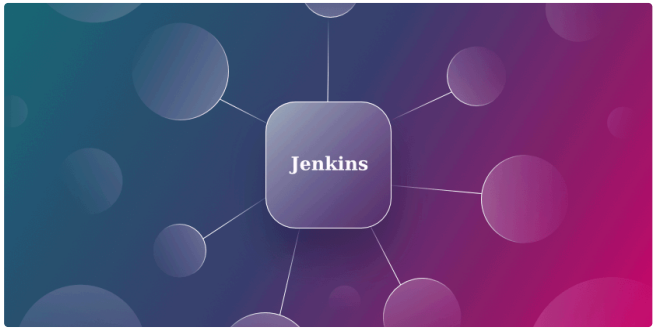
7 Best Practices for Implementing Security Contexts in Kubernetes

Kubernetes security contexts are a fundamental part of safeguarding Kubernet...

3 min read · May 9, 2024

53

Tony



Spacelift in Spacelift

Most Popular Jenkins Alternatives

Streamlit — Next-Gen Web App Development Framework

Usually, before diving into learning a new framework, people typically research what...

✦ · 7 min read · 4 days ago

26 1

Jenkins was a pioneer in the CI/CD space, but today, there are many alternatives available....

9 min read · Apr 22, 2024

9 1

See more recommendations