

**Django + PostgreSQL +Elastic
ON Kubernetes**

Table of contents

1... Glossary

- 1.1... Compute
- 1.2... Storage, Network
- 1.3... Pods Configuration, Infrastructure components

2... Architecture

3... Settings

- 3.1... Settings-Django
- 3.2... Create Dockerfile
- 3.2... Settings-Kubernetes
 - 3.2.1... Helm install
 - 3.2.2... What is Helm?
 - 3.2.3... Create Manifest
- 3.3... Install Manifest
 - 3.3.1... Install Django, PostgreSQL
 - 3.3.2... Install Elastic

3.4... Port-Forward Services

- 3.5... Identify Database
 - 3.5.1... Enter Database
 - 3.5.2... Select Data Table
 - 3.5.3... Select Data
 - 3.5.4... Insert Data
 - 3.5.5... Again Select Data

4... Visualization

- 4.1... Settings Kibana Data
- 4.2... Create Dashboard
 - 4.2.1... Django-Postgre-Count
 - 4.2.2... Log Stream
 - 4.2.3... Metric Memory Pod
 - 4.2.4... Metric CPU Pod

Glossary

Compute



Pod: 파드는 하나 이상의 컨테이너의 그룹이며, 쿠버네티스 애플리케이션의 최소 단위입니다.
같은 포드에 속한 컨테이너끼리 동일한 컴퓨팅 리소스를 공유합니다.



ReplicaSet: 레플리카셋은 클러스터 안에서 움직이는 파드의 수를 유지하는 장치입니다. 클러스터의
파드의 실행을 항상 안정적으로 유지하는 것을 목표로 명시된 파드 개수에 대한 가용성을 보증하는데 사용됩니다.
만약 애플리케이션 오류나 노드 장애 등으로 파드가 정지된 경우 레플리카셋이 자동으로 새로운 파드를
시작합니다.



Deployment: 디플로이먼트는 레플리카셋을 소유하거나 업데이트를 하고, 파드의 선언적인 업데이트와
서버측 롤링 업데이트를 할 수 있는 오브젝트입니다. 배포된 Pod에 대한 rollback을 수행할 수 있습니다.
Pod의 배포되고 update 되는 모든 버전을 추적 할 수 있습니다.
즉 디플로이먼트는 레플리카셋보다 상위 개념이라고 할 수 있습니다.

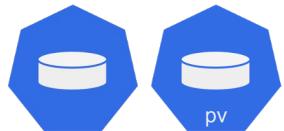


DaemonSet: 데몬셋은 노드가 있는 수만큼 파드를 생성하고 각 노드에 파드를 하나씩 배포합니다. 노드가
다운되어도 데몬셋은 어느 곳에서도 파드를 생성하지 않습니다. 즉, 노드가 클러스터에서 빠졌을 때는 해당
노드에 있던 파드는 그대로 사라질 뿐 다른 곳으로 옮겨가서 실행되거나 하지 않습니다.
주로 사용하는 기능들은 모든 노드에서 로그 수집, 노드 모니터링 데몬 실행

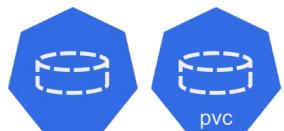


StatefulSet: 스테이트풀셋은 컨텍스트와 내역이 저장되므로 중단되어도 중단된 곳부터 다시 시작할 수 있습니다.
스테이트풀 애플리케이션은 창의 위치, 기본 설정 구성, 최근 활동과 같은 항목을 추적합니다.
주로 Pod의 DNS를 고정시켜주고 싶은 경우나 스토리지가 동일한 상태를 유지하도록 세팅하는 경우에 사용한다.

Storage



PersistentVolume: 퍼시스턴트볼륨(PV)은 볼륨 자체를 의미합니다. 클러스터내에서 리소스로 다뤄집니다. 파드하고는 별개로 관리되고 별도의 생명주기를 가지고 있습니다.

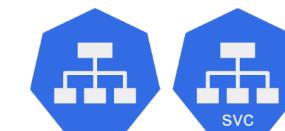


PersistentVolumeClaim: 퍼시스턴트볼륨클레임(PVC)는 사용자가 PV에 하는 요청입니다. 사용하고 싶은 용량이 얼마인지 읽기/쓰기는 어떤 모드로 설정하고 싶은지 등 정해서 요청합니다.



StorageClass: 스토리지클래스는 관리자가 제공하는 스토리지의 "classes"를 설명할 수 있는 방법을 제공한다. 다른 클래스는 서비스의 품질 수준 또는 백업 정책, 클러스터 관리자가 정한 일의의 정책에 매핑될 수 있다. 즉 PV의 크기 및 이름을 정의 할 수 있습니다.

Network



Service: 서비스는 파드들을 통해 실행되고 있는 애플리케이션을 네트워크에 노출시키는 가상의 컴포넌트다. 쿠버네티스 내부의 다양한 객체들이 애플리케이션과, 그리고 애플리케이션이 다른 외부의 애플리케이션이나 사용자와 연결될 수 있도록 도와주는 역할을 한다.

Pods Configuration



ConfigMap: 컨피그맵은 키-값 쌍으로 기밀이 아닌 데이터를 저장하는 데 사용하는 API

오브젝트입니다. 파드는 볼륨에서 환경 변수, 커맨드-라인 인수 또는 구성 파일로 컨피그맵을 사용할 수 있다. 컨피그맵을 사용하면 컨테이너 이미지에서 환경별 구성을 분리하여, 애플리케이션을 쉽게 이식할 수 있습니다.



Secret: 시크릿은 암호, 토큰 또는 키와 같은 소량의 중요한 데이터를 포함하는 오브젝트이다.

이를 사용하지 않으면 중요한 정보가 파드 명세나 컨테이너 이미지에 포함될 수 있다. 시크릿을 사용하는 파드와 독립적으로 생성될 수 있기 때문에, 파드를 생성하고, 확인하고, 수정하는 워크플로우 동안 시크릿(그리고 데이터)이 노출되는 것에 대한 위험을 경감시킬 수 있다.

Infrastructure components



Cluster: 쿠버네티스 클러스터입니다.

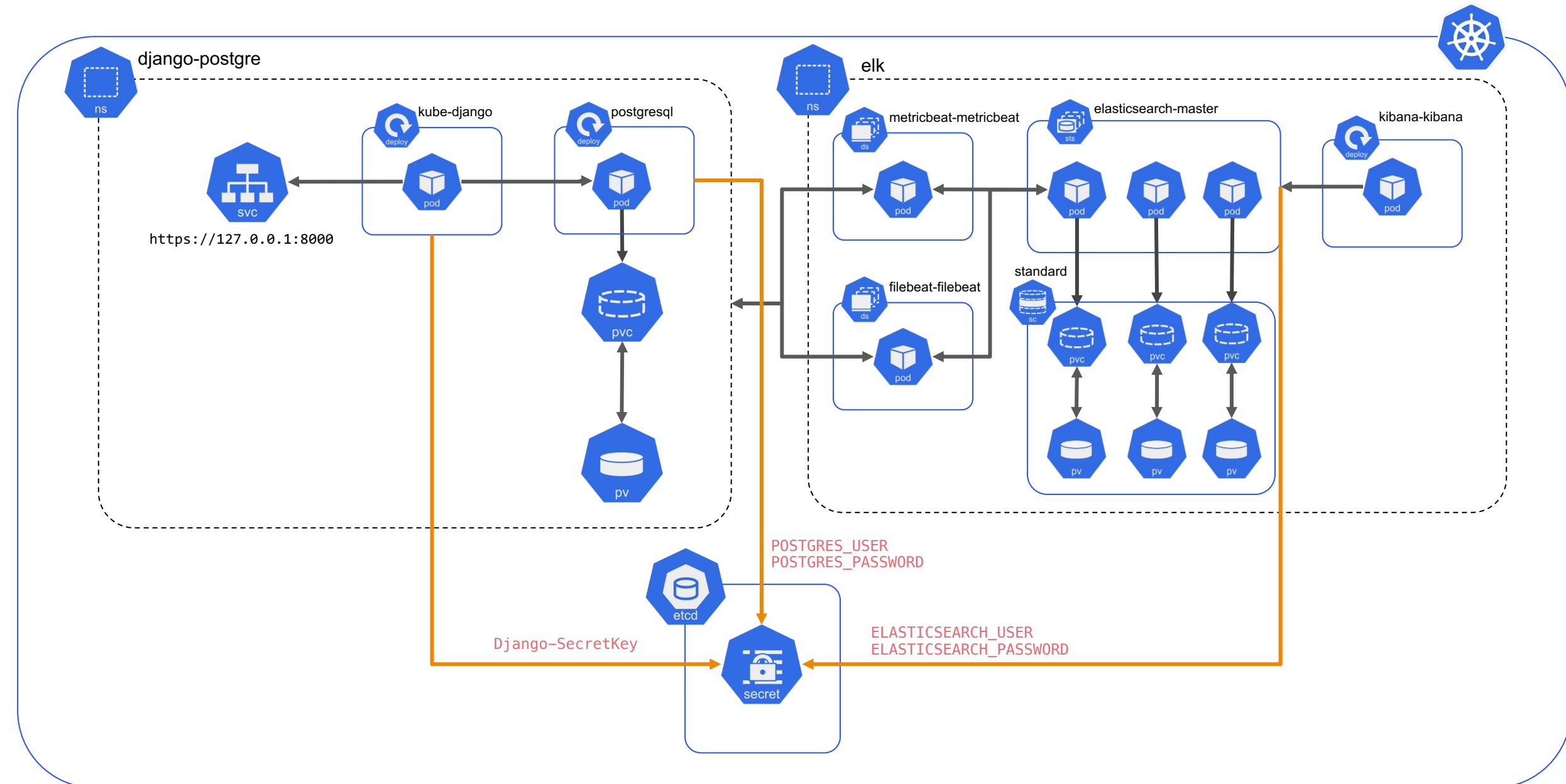


ETCD: etcd는 분산 시스템에서 사용할 수 있는 분산형 키-값 저장소입니다.

고가용성을 위해 여러 곳에 분산되어 동기화되어 있습니다. 또한 쿠버네티스 클러스터의 정보를 저장합니다.

Architecture

Django + PostgreSQL + ELASTIC Kubernetes Architecture



Settings

Settings-Django

Settings-Django



Command

```
1      $ pip3 freeze
asgiref==3.6.0
Django==4.1.4
psycopg2-binary==2.9.5
sqlparse==0.4.3
```

Settings-Django



settings.py

```
ALLOWED_HOSTS = ['*']      # 모든 곳으로부터 접속 허용
:
:
:
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': os.environ.get('POSTGRES_NAME', 'kube-database'),      # DB 이름 추가
        'USER': os.environ.get('POSTGRES_USER', 'root'),      # 사용자 이름 추가
        'PASSWORD': os.environ.get('POSTGRES_PASSWORD', 'toor'),      # 사용자 비밀번호 추가
        'HOST': os.getenv('POSTGRES_SERVICE_HOST', '127.0.0.1'),      # DB IP주소 추가
        'PORT': os.getenv('POSTGRES_SERVICE_PORT', 5432)      # DB 포트번호 추가
    }
}
```

Settings-Django



Command

```
1      $ python3 manage.py createsuperuser
Username (leave blank to use 'root'): root
Email address:
Password:
Password (again):
Superuser created successfully.
```

Settings-Django



	Command
1	\$ python manage.py makemigrations
2	\$ python manage.py migrate

Create Dockerfile



Dockerfile

```
FROM python:latest
# Python 마지막 버전 설치

WORKDIR /code
# Python 컨테이너에 code 디렉토리 생성

COPY kube-django/
# Django 프로젝트 디렉토리를 컨테이너에 복사

RUN pip3 install --upgrade pip
RUN pip3 install -r requirements.txt
# pip list 설치

EXPOSE 8000
# PORT 8000 허용

CMD python3 manage.py runserver 0.0.0.0:8000 --noreload
# 컨테이너 실행시 실행될 명령어
```

Settings-Kubernetes

Helm install



Command

```
1 $ brew install helm # 맥 os  
2 $ choco install kubernetes-helm # 윈도우  
3 $ sudo snap install helm --classic # 리눅스
```

What is Helm?



Helm: Helm이란 쿠버네티스 패키지 매니저입니다. Python의 pip 또는 centOS의 yum과 같은 패키지 매니저입니다. Helm을 사용하면 원하는 패키지들을 쉽게 설치 할 수 있습니다. Helm은 docker hub와 비슷하게 Helm 패키지를 저장하는 [저장소](#)가 있습니다. 따라서 사용자가 원하는 패키지를 직접 설치하면 되며, **values.yaml**로 커스텀을 하여 설치 할 수 있습니다

Create Manifest



elasticsearch.yaml

```
---
```

```
antiAffinity: "soft"
```

```
esJavaOpts: "-Xmx256m -Xms256m"
```

```
# 사용자 PC환경의 맞게 값을 할당하면 된다.
```

```
resources:
```

```
requests:
```

```
cpu: "100m"
```

```
memory: "1024M"
```

```
limits:
```

```
cpu: "1000m"
```

```
memory: "1024M"
```

```
# elasticsearch의 사용자 비밀번호 Default Id는 elastic 이다.
```

```
secret:
```

```
enabled: true
```

```
password: "root"
```

```
# SC 이름을 standard로 하며 PV를 100M씩 할당한다.
```

```
volumeClaimTemplate:
```

```
accessModes: [ "ReadWriteOnce" ]
```

```
storageClassName: "standard"
```

```
resources:
```

```
requests:
```

```
storage: 100M
```

Create Manifest



kube-django.yaml

```
apiVersion: apps/v1
kind: Deployment # 디플로이먼트로 생성한다.
metadata:
  name: kube-django
  namespace: django-postgre # 네임스페이스
  labels:
    app: django
spec:
  replicas: 1
  selector:
    matchLabels:
      app: django
  template:
    metadata:
      labels:
        app: django
    spec:
      containers:
        - name: kube-django
          image: jeff125/kube-django:1.0.4 # Django
          imagePullPolicy: Always
          ports:
            - containerPort: 8000
          env:
            - name: POSTGRES_SERVICE_HOST
              value: Postgre SVC IP를 넣는다.
            - name: POSTGRES_USER # Secrets 항목 이름
              valueFrom:
                secretKeyRef:
                  name: kube-web-secret # Secrets 이름
                  key: POSTGRES_USER # Secrets Key
            - name: POSTGRES_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: kube-web-secret # Secrets 이름
                  key: POSTGRES_PASSWORD # Secrets Key
          resources: #자신 PC환경의 맞게 할당한다.
            requests:
              memory: "64Mi"
              cpu: "250m"
            limits:
              memory: "128Mi"
              cpu: "500m"
          volumeMounts:
            - mountPath: /var/lib/postgresql/data # 데이터가 저장될 위치
              name: postgresql-volume-mount
          volumes:
            - name: postgresql-volume-mount
              persistentVolumeClaim:
                claimName: postgres-pv-claim
```

Create Manifest



kube-django.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: django-svc
  namespace: django-postgre
  labels:
    app: django
spec:
  ports:
    - port: 8000    # Cluster 내부에서 사용될 Service object의 포트입니다.
      targetPort: 8000  # Service object로 들어온 요청을 전달할 target이되는 Pod이 노출하고 있는 포트입니다
      protocol: TCP
    type: NodePort
    selector:
      app: django
```

Create Manifest



postgresql.yaml

```
apiVersion: apps/v1
kind: Deployment # 디플로이먼트로 생성한다.
metadata:
  name: postgresql
  namespace: django-postgre # 네임스페이스
  labels:
    app: postgres-db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres-db
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: postgres-db
        tier: postgreSQL
    spec:
      containers:
        - name: postgresql
          image: postgres
          ports:
            - containerPort: 5432
      env:
        :
      volumeMounts:
        - mountPath: /var/lib/postgresql/data
          name: postgresql-volume-mount
      resources:
        requests:
          memory: "64Mi" # 자신의 PC환경의 맞게 할당한다.
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
      volumes:
        - name: postgresql-volume-mount
          persistentVolumeClaim:
            claimName: postgres-pv-claim
```

Create Manifest



postgresql.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: postgresql-svc
  namespace: django-postgre
  labels:
    app: postgres-db
spec:
  type: ClusterIP
  ports:
  - port: 5432 # Cluster 내부에서 사용될 Service object의 포트입니다.
    targetPort: 5432 # Service object로 들어온 요청을 전달할 target이되는 Pod이 노출하고 있는 포트입니다
    protocol: TCP
  selector:
    app: postgres-db
    tier: postgreSQL
```

Create Manifest



pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgresql-volume-mount
  namespace: django-postgre
spec:
  capacity:
    storage: 100M
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  storageClassName: manual
  persistentVolumeReclaimPolicy: Delete
  hostPath:
    path: /var/lib/postgresql/data
```

```
:
:
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pv-claim
  namespace: django-postgre
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 100M
  storageClassName: manual
```

Create Manifest



secrets.yaml

```
apiVersion: v1
kind: Secret # 시크릿
metadata:
  name: kube-web-secret
  namespace: django-postgre
  resourceVersion: "1"
  type: Opaque
data: # 시크릿의 데이터는 base64로 인코딩 해서
      넣어야 한다.
  POSTGRES_USER: cm9vdA==
  POSTGRES_PASSWORD: dG9vcg==

  :
  apiVersion: v1
  kind: Secret
  metadata:
    name: django-secret-key
    namespace: django-postgre
    resourceVersion: "1"
    type: Opaque
  data:
    SECRET_KEY:
      ZGphbmdvLWluc2VjdXJlLWU0ITl5aGt5IWYxa
      nN2ZGkwNXd6K3clKG8qaGVtcmsxcW0mNnA2Jm
      s5ZHtZGdrXyRq
```

Install Manifest

Install Django, PostgreSQL



Command

```
1 $ kubectl install secrets.yaml -f -namespace=django-postgre  
2 $ kubectl install pv.yaml -f -namespace=django-postgre  
3 $ kubectl install django-postgre.yaml -f -namespace=django-postgre  
4 $ kubectl install postgresql.yaml -f -namespace=django-postgre
```

Install Elastic



Command

```
1      $ helm install elasticsearch elastic/elasticsearch -f elasticsearch.yaml  
      --namespace=elk  
  
2      $ helm install filebeat elastic/filebeat --namespace=elk  
  
3      $ helm install kibana elastic/kibana --namespace=elk  
  
4      $ helm install metricbeat elastic/metricbeat --namespace=elk
```

Port-Forward Services

Port-Forward Services



Command

```
1      $ kubectl port-forward service/elasticsearch-master 9200 --namespace=elk
```

```
Forwarding from 127.0.0.1:9200 -> 9200
```

```
Forwarding from [::1]:9200 -> 9200
```

```
2      $ kubectl port-forward service/kibana-kibana 5601 --namespace=elk
```

```
Forwarding from 127.0.0.1:5601 -> 5601
```

```
Forwarding from [::1]:5601 -> 5601
```

```
3      $ kubectl port-forward service/django-svc 8000 --namespace=django-postgre
```

```
Forwarding from 127.0.0.1:8000 -> 8000
```

```
Forwarding from [::1]:8000 -> 8000
```

Identify Database

Enter Database



Command

```
1      $ kubectl get pods --namespace=django-postgre
NAME                  READY   STATUS    RESTARTS   AGE
kube-django-9b5b768c8-pv8kn   1/1     Running   0          30m
postgresql-6cdb4849c9-xlpr8   1/1     Running   0          30m

2      $ kubectl exec -it postgresql-6cdb4849c9-xlpr8 -n django-postgre /bin/bash
root@postgresql-6cdb4849c9-xlpr8:#
root@postgresql-6cdb4849c9-xlpr8:~# psql -U root -d kube-database
psql (15.1 (Debian 15.1-1.pgdg110+1))
Type "help" for help.

kube-database=#

```

Select Data Table



Command

```
kube-database=# \dt
```

List of relations

Schema	Name	Type	Owner
public	auth_group	table	root
public	auth_group_permissions	table	root
public	auth_permission	table	root
public	auth_user	table	root
public	auth_user_groups	table	root
public	auth_user_user_permissions	table	root
public	django_admin_log	table	root
public	django_content_type	table	root
public	django_migrations	table	root
public	django_session	table	root
(10 rows)			

Select Data



Command

```
kube-database=# SELECT * FROM auth_user;

id | password | last_login | is_superuser | username | first_name
| last_name | email | is_staff | is_active | date_joined
---+-----+-----+-----+-----+-----+
+-----+-----+-----+
1 | pbkdf2_sha256$390000$GE4OwDDBhLmGXdh6lOtTCI$rt6m/9O19uGklli8E3d2UMjxVtllEi7S02ipW4UqZNw= | 2023-01-06 05:37:19.532849+00 | t | root |
| t | t | 2022-12-29 07:57:07.289251+00
(1 row)
```

Insert Data

Django 관리

환영합니다, ROOT. 사이트 보기 / 비밀번호 변경 / 로그아웃

홈 · 인증 및 권한 · 사용자(들) · 사용자 추가

필터에 타이핑 시작...

인증 및 권한

그룹 [+ 추가](#)

사용자(들) [+ 추가](#)

사용자 추가

첫 번째로, 사용자명과 비밀번호를 입력하세요. 그 후, 독자는 더 많은 사용자 옵션을 수정할 수 있습니다.

사용자 이름:

150자 이하 문자, 숫자 그리고 @/./+/-/_만 가능합니다.

비밀번호:

다른 개인 정보와 유사한 비밀번호는 사용할 수 없습니다.
비밀번호는 최소 8자 이상이어야 합니다.
통상적으로 자주 사용되는 비밀번호는 사용할 수 없습니다.
숫자로만 이루어진 비밀번호는 사용할 수 없습니다.

비밀번호 확인:

확인을 위해 이전과 동일한 비밀번호를 입력하세요.

[저장 및 다른 이름으로 추가](#) [저장 및 편집 계속](#) [저장](#)

<https://localhost:8000/admin/auth/user/add>

Again Select Data



Command

```
kube-database=# SELECT * FROM auth_user;

id |          password          |      last_login      | is_superuser | username | first_name |
last_name | email | is_staff | is_active |      date_joined

----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+
1 | pbkdf2_sha256$390000$GE4OwDDBhLmGXdh6lOtTCI$rt6m/9O19uGklli8E3d2UMjxVtllEi7S02ipW4UqZNw= | 2023-01-06 05:37:19.532849+00 | t     | root   |      |
| t     | t     | 2022-12-29 07:57:07.289251+00
6 | pbkdf2_sha256$390000$tj8DCNFxT6sI1Wgui799y8$XKV9zxDesW1cu30mxBjrz8mQvyPJeeNK+BZ+d3hwfCY= | f     | test   |      |
| f     | t     | 2023-01-06 06:57:37.444967+00
(2 rows)
```

Visualization

Settings Kibana Data

The screenshot shows the Elastic Kibana home page. At the top, there's a navigation bar with the elastic logo, a search bar, and user icons. Below it, a "Welcome home" section features four colored tiles: yellow for "Enterprise Search", pink for "Observability", teal for "Security", and blue for "Analytics". Each tile has an icon and a brief description. Below this, a "Get started by adding integrations" section includes a "Add integrations" button and links to "Try sample data" and "Upload a file". To the right is a decorative graphic of a house with various charts and graphs floating around it. At the bottom, a "Management" section contains four cards: "Manage permissions", "Monitor the stack", "Back up and restore", and "Manage index lifecycles". The "Stack Management" card is highlighted with a blue border.

<http://localhost:5601/app/home/>

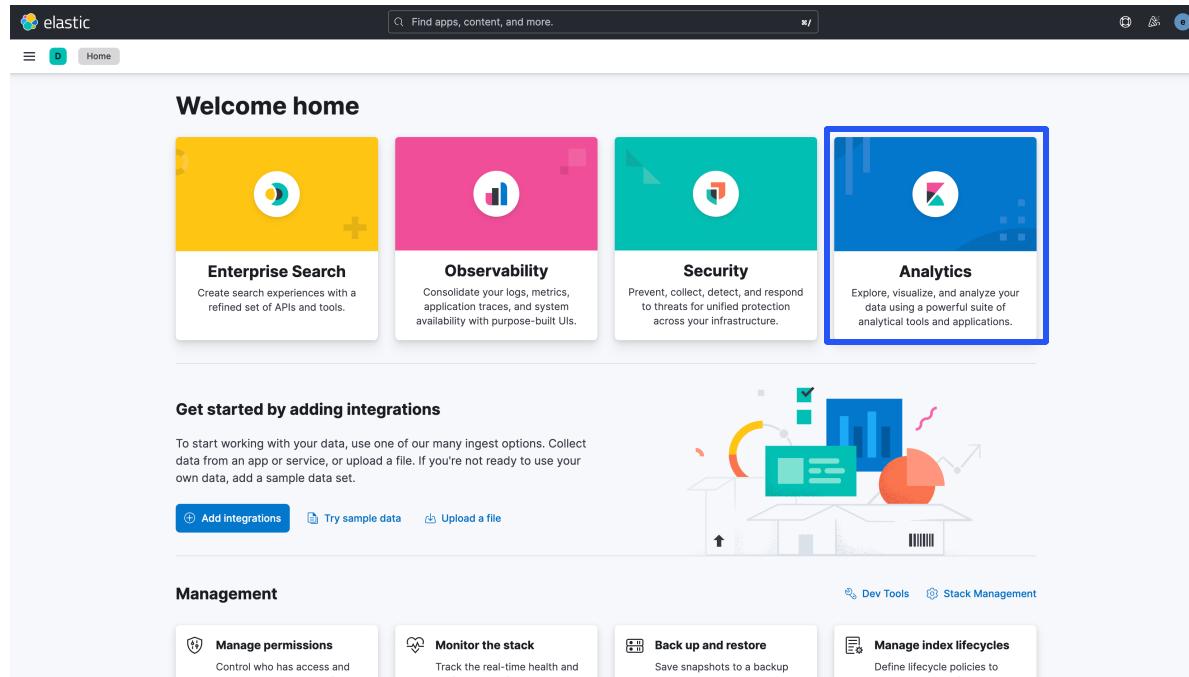
Settings Kibana Data

The screenshot shows the Elasticsearch Stack Management interface with the 'Data views' tab selected. On the left, a sidebar lists various management options like Index Management, Security, and Kibana. The main area is titled 'Create data view' and contains fields for 'Name' (set to 'filebeat and metricbeat') and 'Index pattern' (set to 'filebeat-* and metricbeat-*'). A note below the index pattern field specifies: 'Enter an index pattern that matches one or more data sources. Use an asterisk (*) to match multiple characters. Spaces and the characters , /, ?, <, >, | are not allowed.' Below these fields is a dropdown for 'Timestamp field'. A link 'Show advanced settings' is visible. At the bottom right is a blue button labeled 'Save data view to Kibana'. The top navigation bar includes the elastic logo, a search bar, and other navigation links.

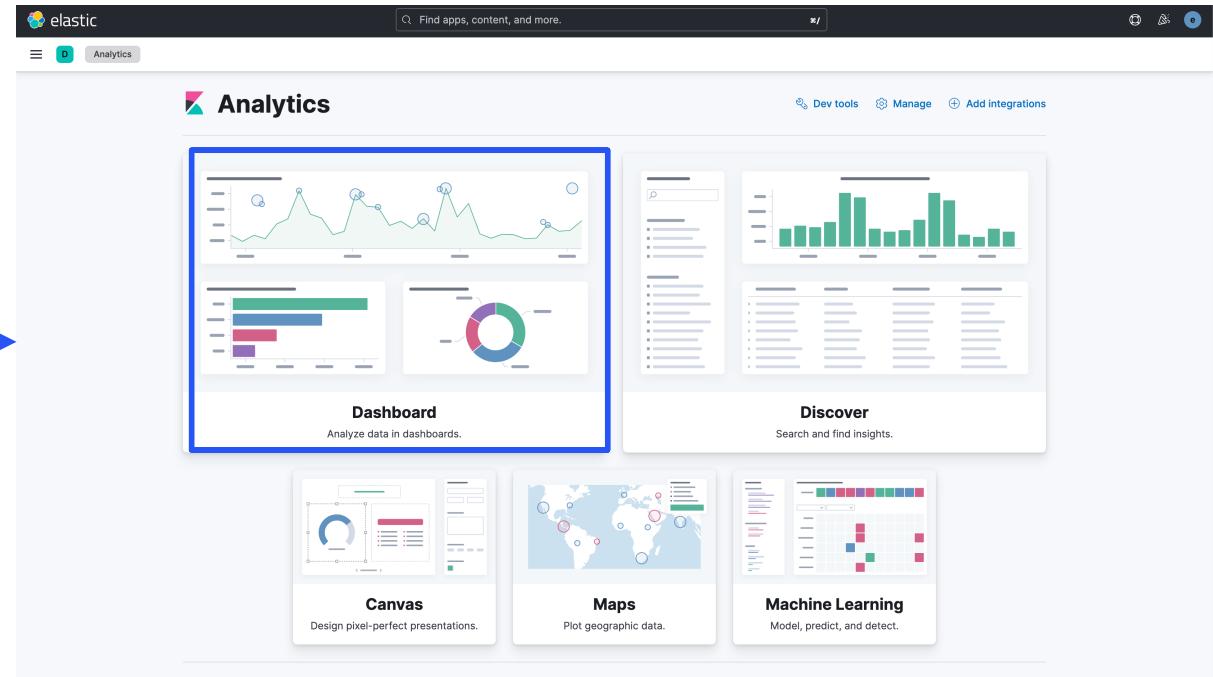
<http://localhost:5601/app/management/kibana/dataViews>

Create Dashboard

Create Dashboard

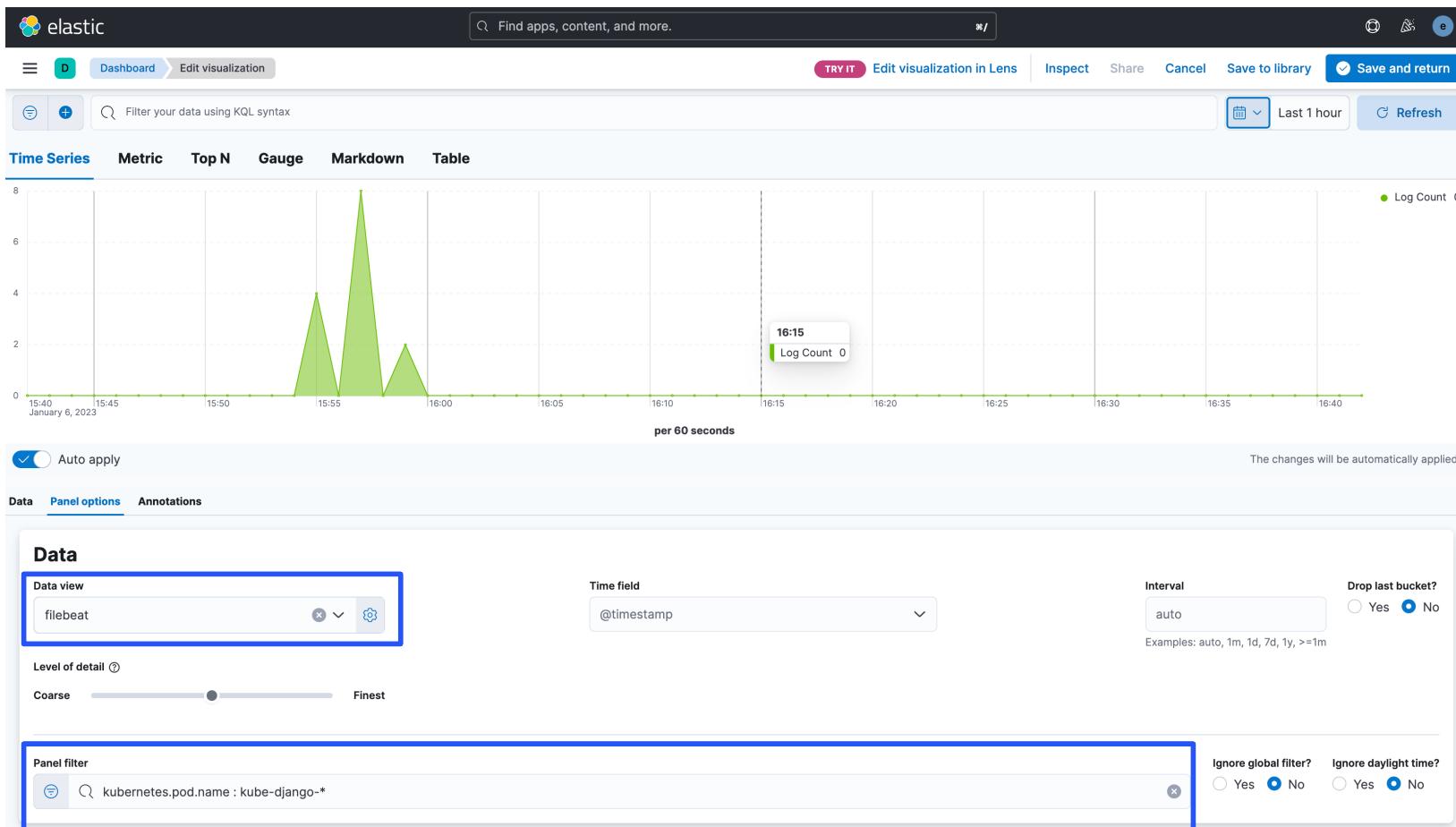


The screenshot shows the Elastic Home page. At the top, there's a navigation bar with the Elastic logo, a search bar, and user profile icons. Below the header, a "Welcome home" section features four cards: "Enterprise Search" (yellow), "Observability" (pink), "Security" (teal), and "Analytics" (blue). The "Analytics" card is highlighted with a blue border. Below this, a "Get started by adding integrations" section contains a central illustration of data flowing into a central hub, along with buttons for "Add integrations", "Try sample data", and "Upload a file". At the bottom, there's a "Management" section with links for "Manage permissions", "Monitor the stack", "Backup and restore", and "Manage index lifecycles".

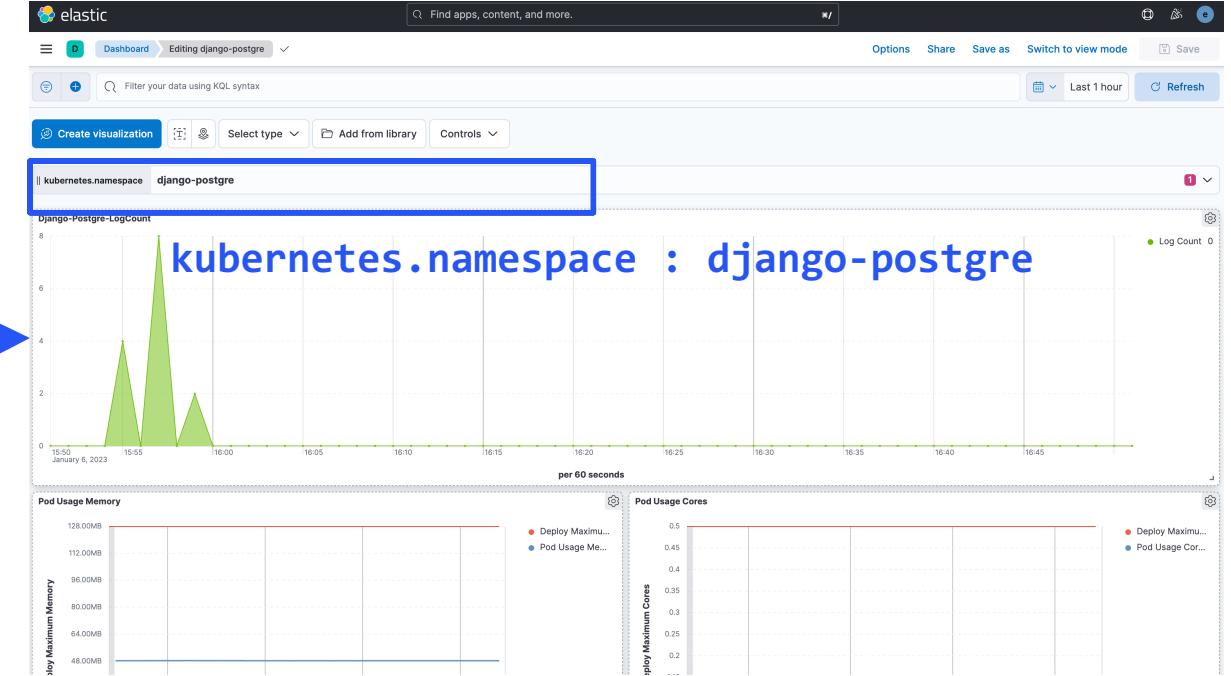
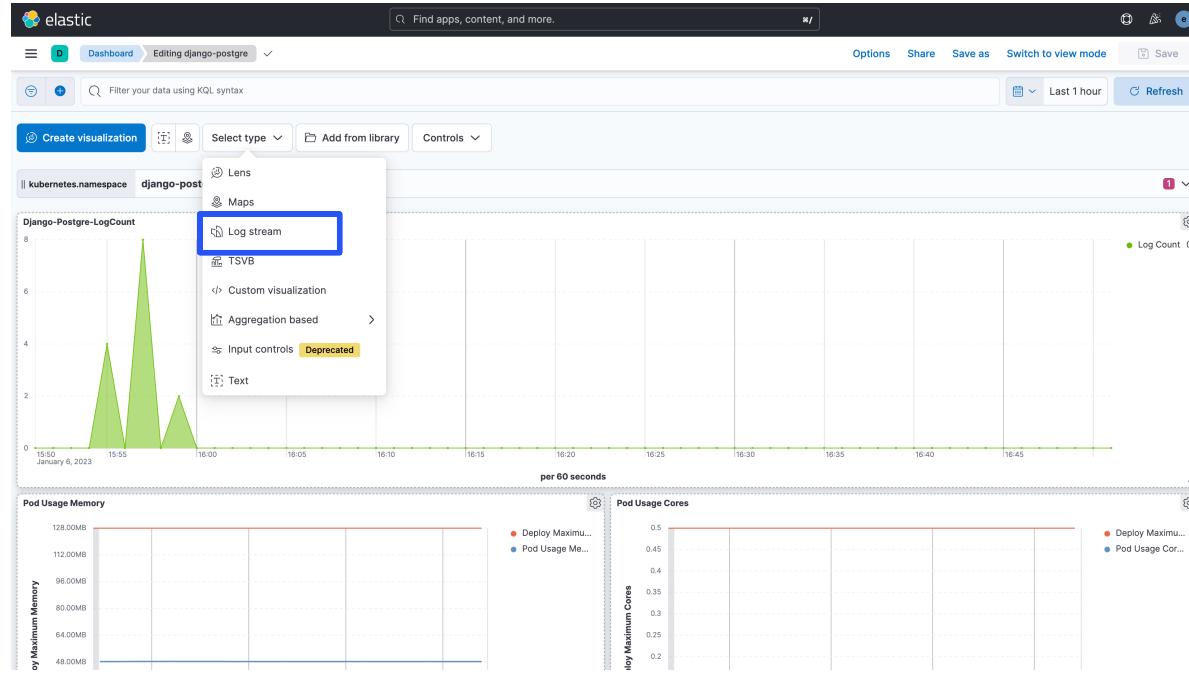


The screenshot shows the Elastic Analytics page. At the top, there's a navigation bar with the Elastic logo, a search bar, and user profile icons. Below the header, a "Analytics" section features several cards: "Dashboard" (blue, highlighted with a blue border), "Discover", "Canvas", "Maps", and "Machine Learning". The "Dashboard" card contains sub-sections for "Analyze data in dashboards" and "Discover". At the bottom, there are links for "Dev tools", "Manage", and "Add integrations".

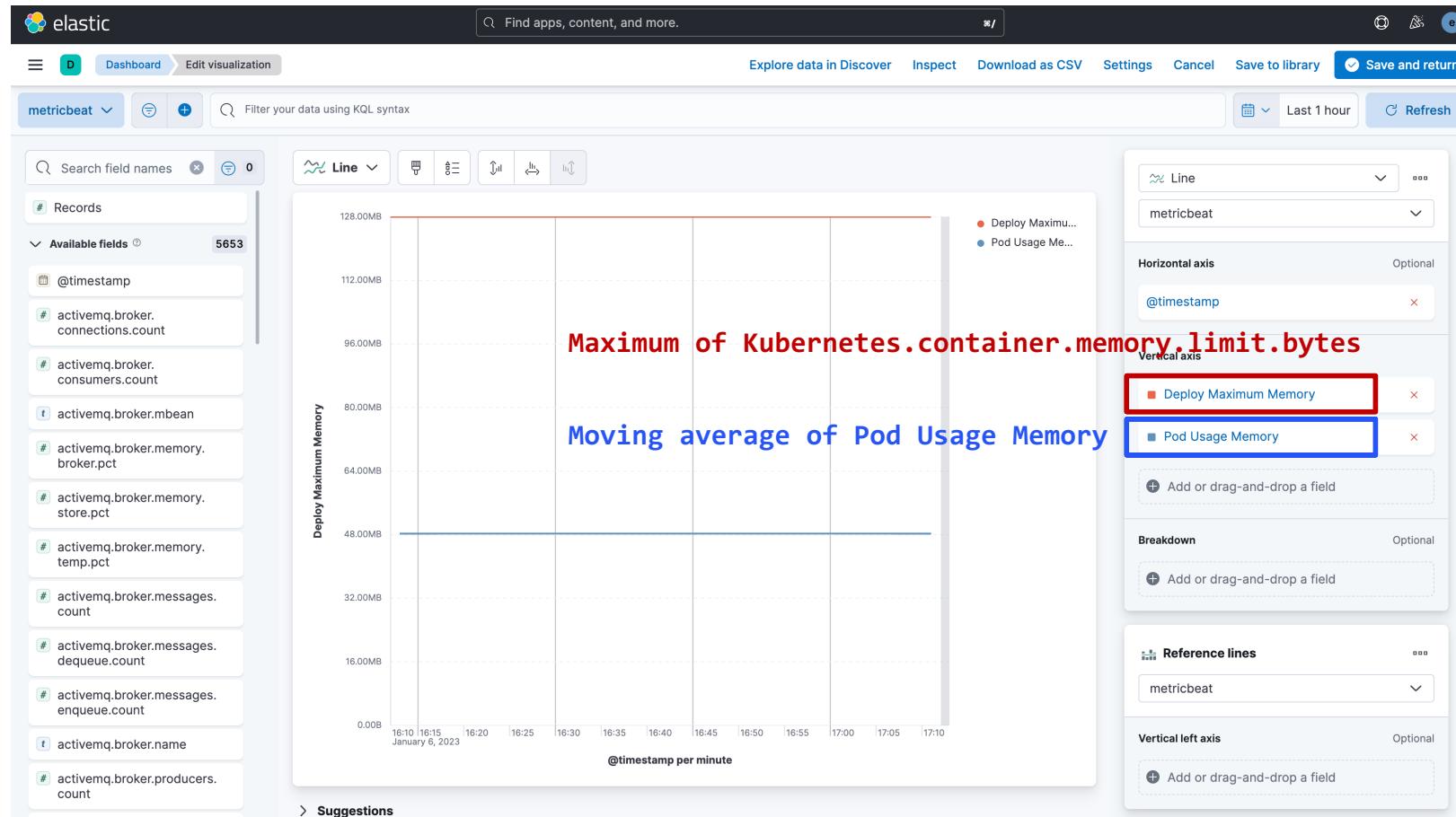
Django-Postgre-LogCount



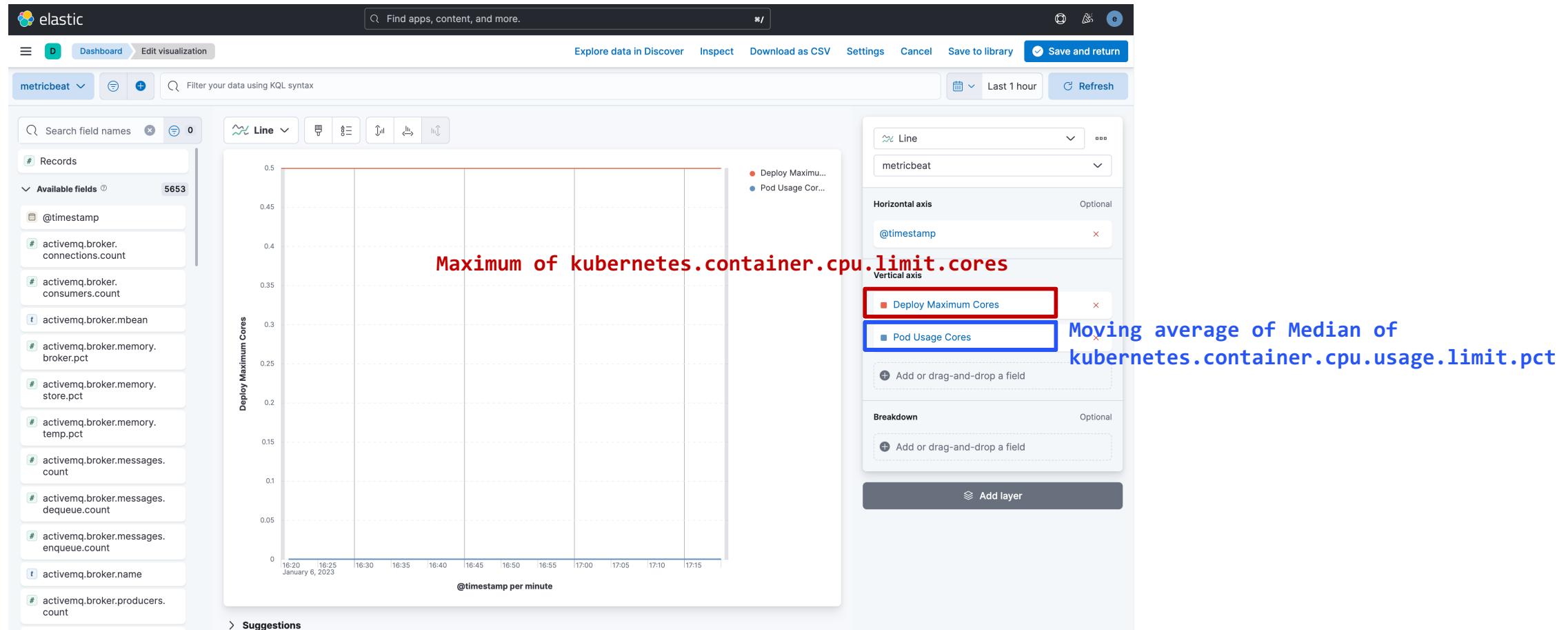
Log Stream



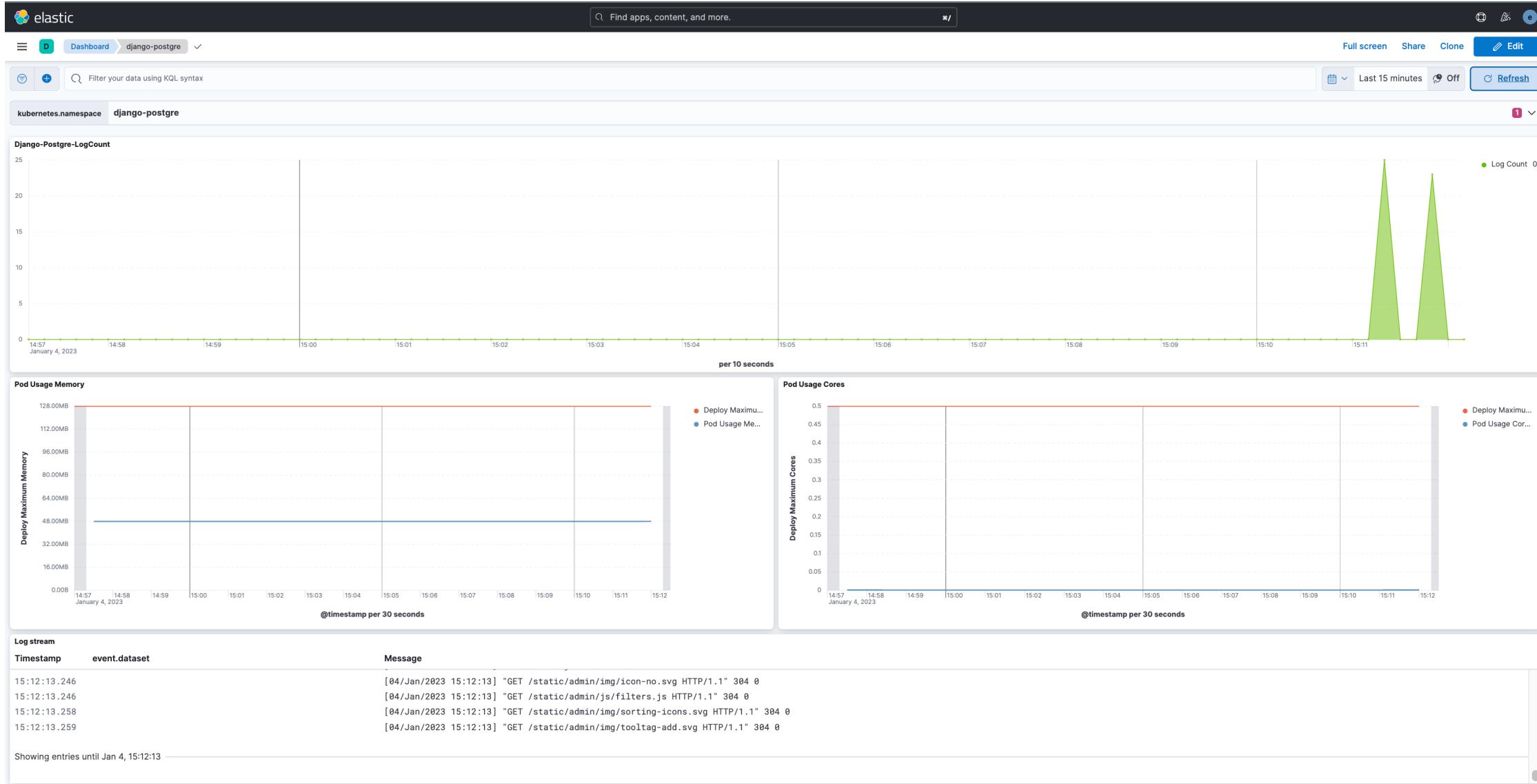
Metric Memory Pod



Metric CPU Pod



Dashboard



쿠버네티스 (minikube)를 활용한 Cloud DevOps에 대해 전반적으로 알아가는 시간을 가졌습니다.

실제 GCP, AWS, Azure와 같은 클라우드 플랫폼과 직접 연동하지는 못했지만, 공부하면서 각 플랫폼마다 Document가 자세하게 설명되었으며, 그 설명을 참고하면서 제작하기도 했습니다.

몇 가지 아쉬운 점이 있는데 실제 클라우드 환경이 아니라 로컬에서 작업 함으로써 resource 메모리 부족 문제이 가장 문제였습니다. 그래서 다양한 컨테이너를 활용못해서 아쉬웠습니다.