## Objective:

Develop a REST API that computes and returns the nth number in the Fibonacci sequence. Your task is to not only implement the functionality but also consider how this service could be deployed in a production environment.

## Task Description:

* Requirements:
  + The service should accept an integer n as input and return the nth Fibonacci number. The Fibonacci sequence is defined as F(0) = 0, F(1) = 1, and F(n) = F(n-1) + F(n-2) for n > 1. [Eg: n=5, Fibonacci Series [0,1,1,2,3,5]
    - Example Responses:
      * Input: The API receives a GET request with a parameter n set to 2.
      * Output: The API should respond with 1, as F(2) = 1.
      * Input: The API receives a GET request with a parameter n set to 10.
      * Output: The API should respond with 55, as F(10) = 55.

**Framework Used**: Python with Flask

**Step 1: Python Script using Flask framework**

GitHub link for accessing the files:

<https://github.com/lisaelsa/Fibonacci/tree/main>

Prerequisites to run the Fibonacci.py file in your machine:

* Install Python
* Install Flask

To run the script: **python Fibonacci.py**

API to access: <http://127.0.0.1:5000/fibonacci?n=3> [Make sure 5000 port is free]

**Step 2: Containerization**

Technology used: Docker

Prerequisites: Install Docker on your machine or Docker Desktop can be used

To build the docker image: [Already done and the final image can be used for testing]

* Clone the gihub repository: git clone <https://github.com/lisaelsa/Fibonacci.git>
* Run: docker build -t <tagname> .

***Note: Make sure docker is installed in the machine***

**Docker Image Link** **for testing**: <https://hub.docker.com/repository/docker/lisaelsa/fibonacciapi/general>

To test the docker container:

* Pull image to your machine: docker pull lisaelsa/fibonacciapi:latest
* Run docker image: docker run -d -p 5000:5000 lisaelsa/fibonacciapi
* API to access: <http://127.0.0.1:5000/fibonacci?n=3> [Make sure 5000 port is free]

**Step 3: Continuous Integration/Continuous Deployment (CI/CD) processes**

CI/CD

**Jenkins Azure DevOps Gitlab**

**(CI/CD) (Cloud Service) (SCM + CI/CD)**

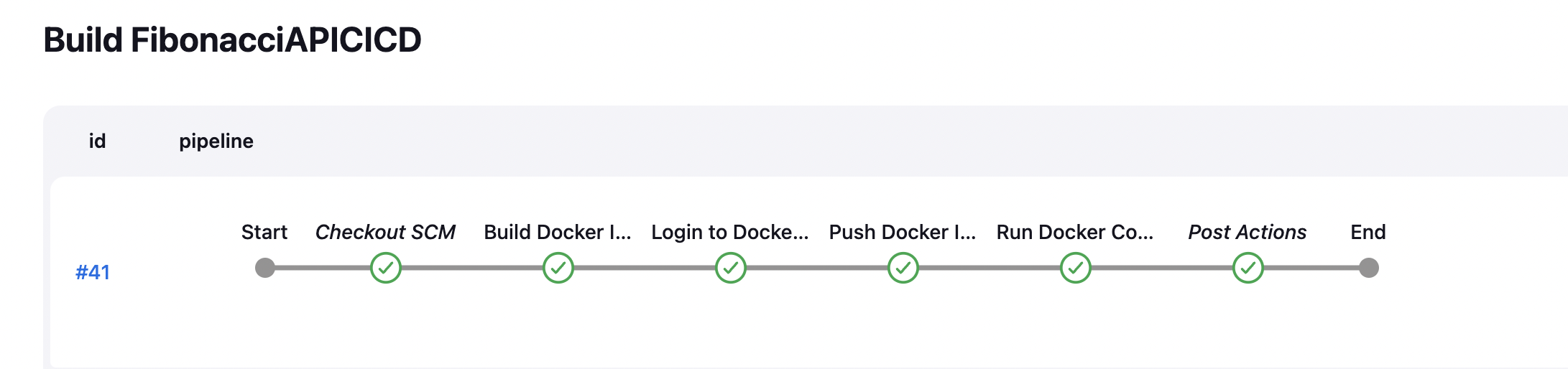
**Implementation**

* Jenkins (Open Source Automation server)
* Azure DevOps (Microsoft Product)

***CI/CD Implementation using Jenkins:***

***Steps:***

* Installation of Jenkins
* Download necessary plugins
* Configuring agent/node
* Manage your dockerhub/registry credentials
* Create a Jenkins job for enabling CI/CD of our Fibonacci application
* Jenkins CI/CD Job: <http://localhost:8080/job/FibonacciAPICICD/>
* Build Triggers: Poll SCM
* Github Project URL used: <https://github.com/lisaelsa/Fibonacci.git/>
* Pipeline Mode: Pipeline Script from SCM using Jenkinsfile
* Pipeline Stages:



API to access: <http://127.0.0.1:5000/fibonacci?n=3>

***CI/CD Implementation using Azure DevOps:***

***Steps:***

* Sign into Azure Portal and Azure Devops

<https://dev.azure.com/lisaelsa1996>

<https://portal.azure.com/#home>

* Create Resource Group, ACR
* Configure Self-hosted agent
* Create CI Pipeline
* Link: <https://dev.azure.com/lisaelsa1996/FibonacciAPI/_build?definitionId=2>
* Enable CI under triggers
* Create Azure Web App for deploying the Docker container
* Deployment Slots
* Deploy multiple version of our app to different slots or environments [Dev,Stage,Production]
* Safe testing environment before deploying to production
* Rollback to previous versions of the app
* Deployment center where shows the source
* Deployment slots [By default prod, stag can be additionally added n swap of the areas can be done]
* Create Release Pipeline
* Link: <https://dev.azure.com/lisaelsa1996/FibonacciAPI/_releaseDefinition?definitionId=1&_a=environments-editor-preview>
* Enable the trigger to create a new release every time a new build is available

**Step 3: Scaling**

**Scaling**

**Kubernetes Openshift Azure Web App**

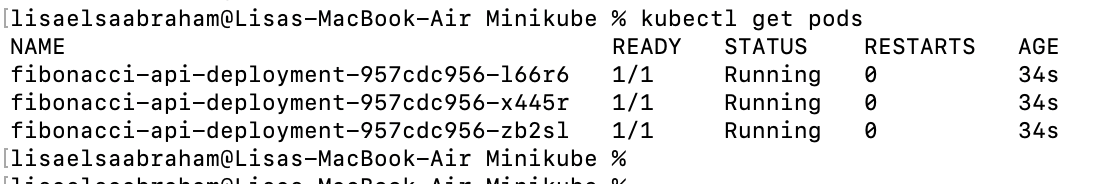
**Implementation**

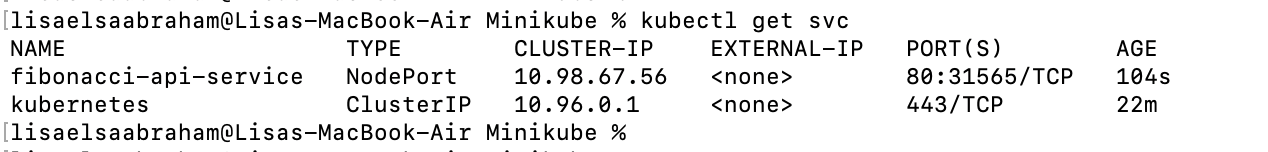
* Kubernetes
* Azure Web App

***Scaling Using Kubernetes***

***Steps:***

* Install Kubernetes on your machine.
* Write Deployment file for FibonacciAPI App mentioning the replicas required
* Apply Kubernetes file and check for pods and services running
* Different Scaling methods
* HPA: Scales pods based on resource utilization metrics
* VPA: Adjust Resource Requests and limits for pods
* Cluster Autoscaler: Adjusts the number of nodes in the cluster based on pod scheduling needs

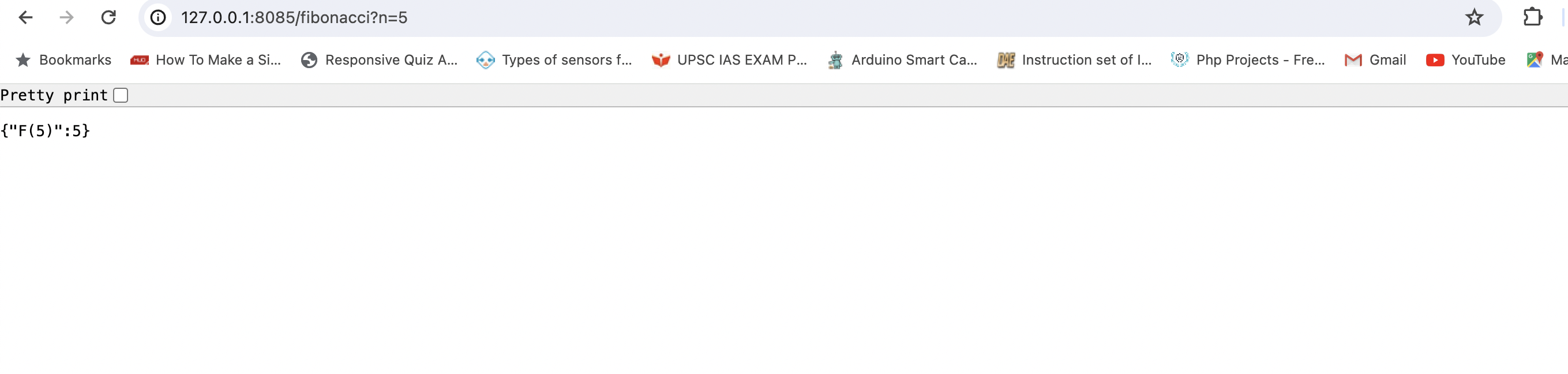




To access the Fibonacci api: kubectl port-forward svc/fibonacci-api-service 8085:80



**NodePort Service:** [**http://127.0.0.1:8085/fibonacci?n=5**](http://127.0.0.1:8085/fibonacci?n=5)

****

***Scaling Using Azure WebApp:***

We can enable autoscaling to automatically adjust the number of instances (horizontal scaling) based on demand. We can define scaling rules based on metrics like CPU utilization, memory usage, or request count.

This feature is enabled for Standard or Premium tier.

If horizontal scaling is not enough to handle the increased load, then we need to upgrade the pricing tier of our Azure Web App to a higher tier with more resources (vertical scaling)

Web app-> [Scale up/ scale Out]. We can either manually do the scaling or mention the scale rule where we mention the metrics upon which the scaling to be done such as CPU%, date schedule

**Step 4: Monitoring**

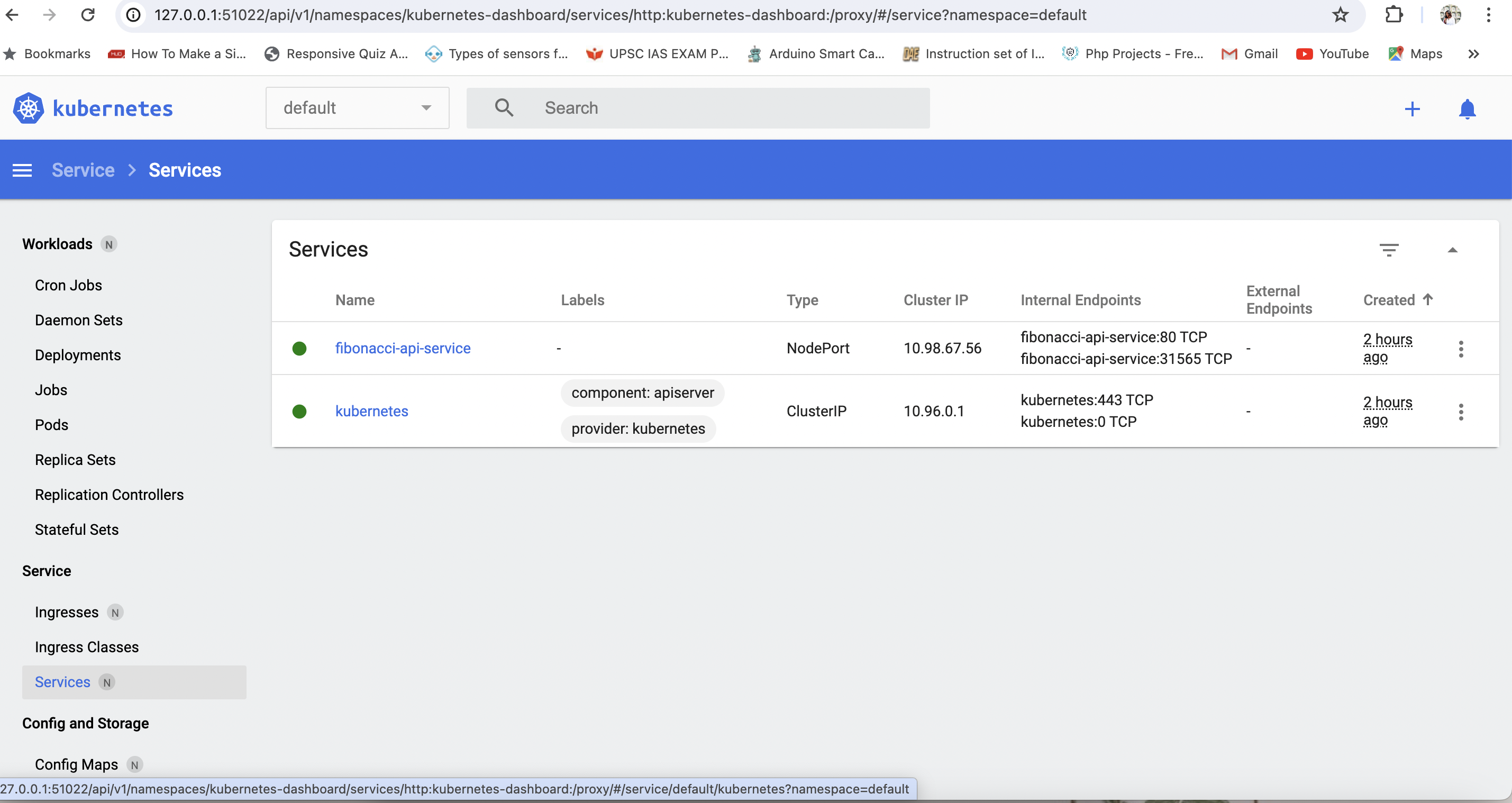
**Azure Devops**

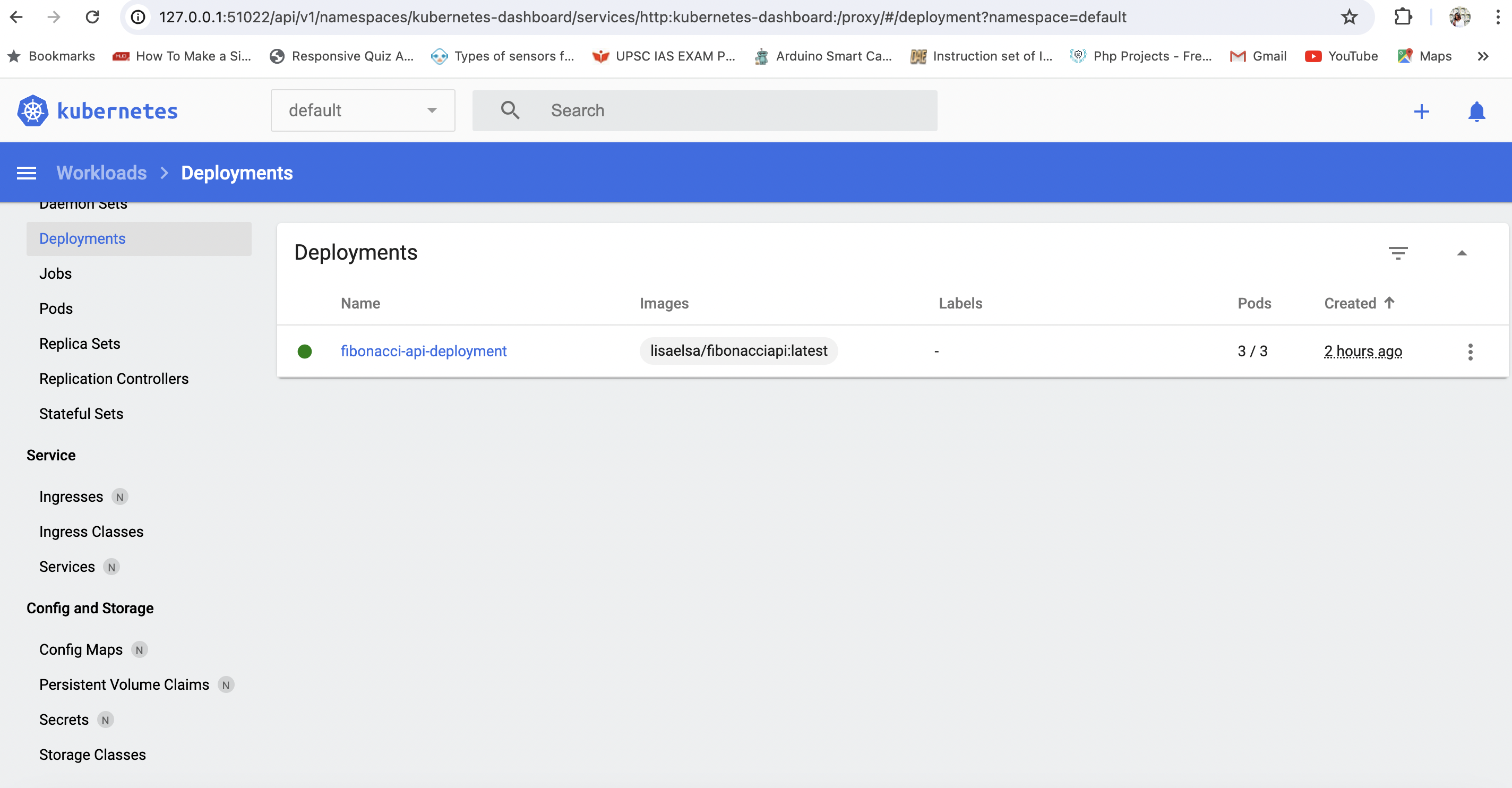
Azure Monitor application insights is an Application Performance Management (APM) service for developers and DevOps professionals. Enable it while creating web app to automatically monitor our application. It will detect performance anomalies, and includes powerful analytics tools to help us diagnose issues and to understand what users actually do with our app.

While creating web app for Fibonacci API, I enabled the Application Insights under monitoring section. PFB the Snapshot of Application Insights generated for Fibonacci App deployed using web app.

**Kubernetes**

[**http://127.0.0.1:51022/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/#/service?namespace=default**](http://127.0.0.1:51022/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/#/service?namespace=default)

****

****

**Best Monitoring Tools:** Kiali, Prometheus, Grafana