ElasticSearch and Kibana deployment

using Microservices

In today’s business environment, enterprises must respond to client needs and changing conditions more rapidly than ever. To keep up, software applications must be quick to deploy, easy to maintain, and always available.

While traditional architecture can still handle a lot of this, there is a limit. At some point, a more dynamic, scalable approach to application development can become critical to the future of the business.

One such approach is a microservice architecture. Microservices promise quicker and easier software changes compared to traditional monolithic architectures by modularizing complex applications.

Developers then compose applications from the resulting interchangeable, upgradable, and scalable parts. In an ideal world, this modular architectural style accelerates business growth by enabling the agile deployment of innovative functionality.

Microservices – or microservices architecture – are applications that are arranged or structured as a collection of loosely coupled services. In general, microservices have these characteristics:

* Each microservice has its own data model and manages its own data.
* Data moves between microservices using “dumb pipes” such as an event broker and/or a lightweight protocol like REST.
* Small scope that encompasses a single piece of business functionality
* Internal operations are a “black box”, accessible to external programs only via API

**The Advantages of Microservices:**

1. Microservices work well with agile development processes and satisfy the increasing need for a more fluid flow of information.
2. Microservices are independently deployable and allow for more team autonomy

* Each microservice can be deployed independently, as needed, enabling continuous improvement and faster app updates.
* Specific microservices can be assigned to specific development teams, which allows them to focus solely on one service or feature. This means teams can work autonomously without worrying what’s going on with the rest of the app.

1. Microservices are independently scalable.

* As demand for an app increases, it’s easier to scale using microservices. You can increase resources to the most needed microservices rather than scaling an entire app. This also means scaling is faster and often more cost-efficient as well.

1. Microservices reduce downtime through fault isolation.

* If a specific microservice fails, you can isolate that failure to that single service and prevent cascading failures that would cause the app to crash. This fault isolation means that your critical application can stay up and running even when one of its modules fails.

1. The smaller codebase enables teams to more easily understand the code, making it simpler to maintain.

* Microservice typically have small codebases, making them easier to maintain and deploy. It’s also much easier to keep the code clean and for teams to be wholly responsible for specific services.

**The Disadvantages of Microservices**

1. Microservices create different types of complexity than monolithic applications for development teams.

* First, communication between services can be complex. An application can include dozens or even hundreds of different services, and they all need to communicate securely.
* Second, debugging becomes more challenging with microservices. With an application consisting of multiple microservices and with each microservice having its own set of logs, tracing the source of the problem can be difficult.
* And third, while unit testing may be easier with microservices, integration testing is not. The components are distributed, and developers can’t test an entire system from their individual machines.

1. Interface control is even more critical.

* Each microservice has its own API, which apps rely on to be consistent. While you can easily make changes to a microservice without impacting the external systems interacting with it, if you change the API (the interface), any application using that microservice will be affected if the change is not backwards compatible.
* A microservices architecture model results in many APIs, all crucial to the operation of the enterprise — so interface control becomes mission-critical.

1. Up-front costs may be higher with microservices.

* For microservices architecture to work for your organization, you need sufficient hosting infrastructure with security and maintenance support, and you need skilled development teams who understand and manage all the services.

**Deploy ElasticSearch**

These guidelines apply to any Linux operating system that is configured to use Docker.

1. Deploy a data storage block

docker volume create elasticsearch\_data

1. Deploy a configuration storage block

docker volume create elasticsearch\_conf

1. Pull the image from the repository

docker pull docker.elastic.co/elasticsearch/elasticsearch:7.15.2

1. Launch the service

docker run -d -p 127.0.0.1:9200:9200 \

--name=elasticsearch \

--hostname=elasticsearch \

--restart=always \

-v elasticsearch\_data:/usr/share/elasticsearch/data \

-v elasticsearch\_conf:/etc/elasticsearch \

-v elasticsearch\_conf:/usr/share/elasticsearch/config \

docker.elastic.co/elasticsearch/elasticsearch:7.15.2

**Deploy Kibana**

1. Deploy a data storage block

docker volume create kibana\_data

1. Pull the image from the repository

docker pull docker.elastic.co/kibana/kibana:7.15.2

1. Launch the service

docker run -d -p 127.0.0.1:15601:15601 \

--name=kibana \

--hostname=kibana \

--restart=always \

-v kibana\_data:/usr/share/kibana/config \

-e "SERVER\_BASEPATH=/kibana" \

docker.elastic.co/kibana/kibana:7.15.2

**Deploy NginX**

1. Deploy a storage block

docker volume create nginx\_conf

docker volume create nginx\_data

1. Pull the image from the repository

docker pull nginx:alpine

1. Launch the service

docker run -d -p 80:80 -p 443:443 \

--name=nginx \

--hostname=nginx \

--network=localnet \

--restart=always \

-v nginx\_conf:/etc/nginx \

-v nginx\_data:/var/www/html \

nginx:alpine

Microservices architecture advantages and disadvantages differ greatly from traditional monolithic architecture, and this model isn’t ideal for every organization. However, the big shift to this modular architectural style is happening for a reason — more enterprises are realizing the need for faster, easier, more agile application development, and microservices enable this in ways monolithic architecture simply cannot.