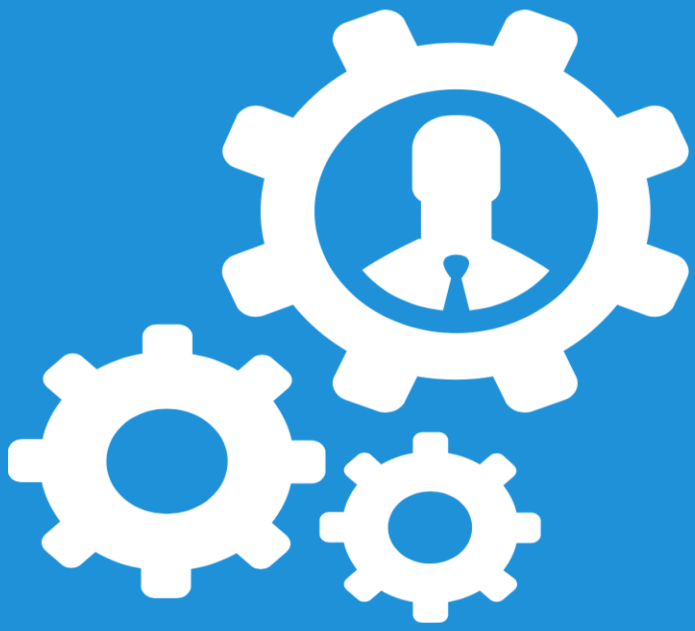
Microservices for DevOps

The Fundamental Understanding Of Addressing Microservices Concern



By Anish Nath

Microservices For DevOps

Anish Nath

This book is for sale at http://leanpub.com/microops

This version was published on 2020-03-24

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

About

2 MINUTE READ

Microservices for DevOps is written on pure DevOps style. This book contains a hands-on practical example of how you can convert your traditional web server hosted on your VM to Monolithic architecture to Microservices pattern.

DevOps is considered as another primary benefit of Microservices and in the context of services that can be deployed independently and frequently

The relationship between Microservices and DevOps is that Microservices architectures *require* DevOps to be successful. if you are coming from Services Oriented Architecture background you can relate many concepts that are generic to Microservices.

While Monolithic applications have many drawbacks, they have the benefit of being a single application that is not a complex distributed system with multiple moving parts and independent tech stack.

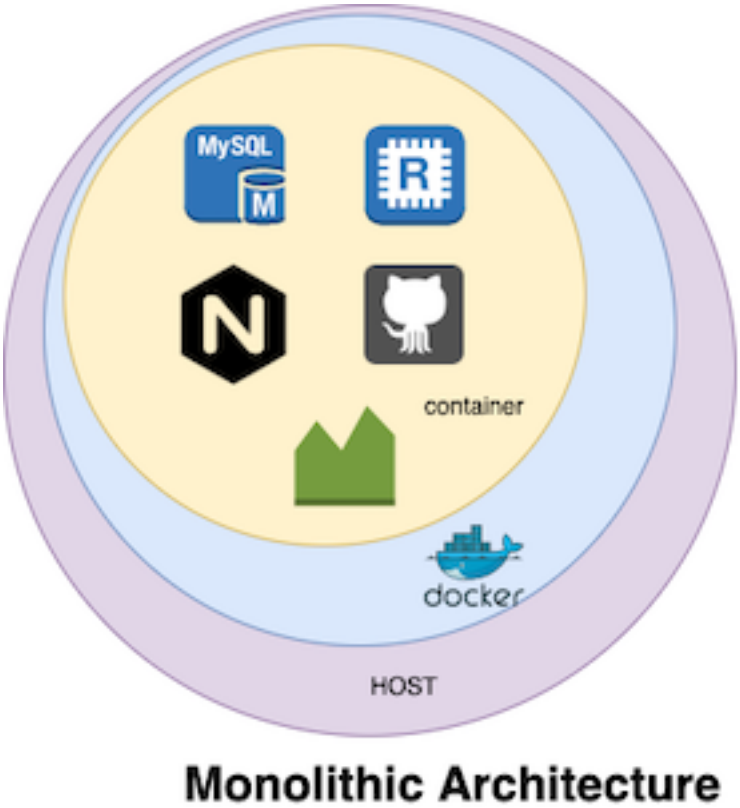
SPOILER: Monolithic Architecture: I love it because it's less time consuming to develop and code can go to the production very fast. Why Microservices ?? I can get lost easily

Concepts

4 MINUTE READ

What is Monolithic Architecture ??

The dictionary definition **monolithic** is: formed of a single large block of stone. In Docker terminology, a Docker image builds with many deployment like Apache, Nginx, Redis, MySQL, etc running on a single container as shown in the diagram below.



Monolithic System

This kind of architecture is still in practice with may DevOps developer around the world and it works very very well when time matter and on dirty PoC.

There are many advantages of this Monlothihc architecture

- Budget-Friendly

- Easy to Control
- Easy to manage as having less Configuration file
- Messy Code [Many developers find it easy to read messy code Instead of Modern Modular code Structure]

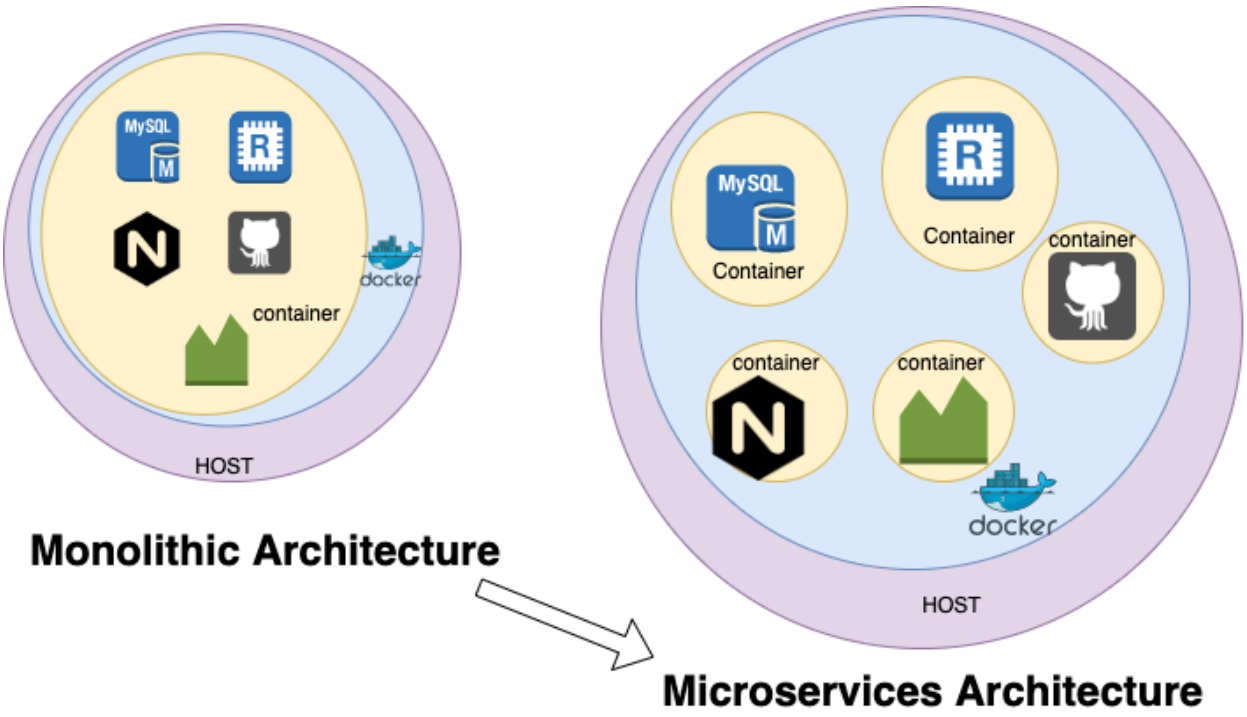
Don't forget about the disadvantages also

- Hard to scale
- Security patching of Node wise is tough
- Uptime Issues when new deployment comes in.
- Managing PIPLE line

What is Microservices Architecture ??

SPOILER ALERT Microservices sucks ?? good in talking but when it comes to production I prefer Monlothihc architecture and make that perfect.

The **monolithic** is: formed of a single large block of stone, now it's time to break that stone in smaller pieces, in DevOps terminology they are called Services



MicroServices

As shown in the above diagram, in microservices architecture one container is exactly hosting one services

- A separate container for MySQL
- A Separate Container for Redis
- A Separate Container for Nginx
- A Separate Container for Tomcat

Unlike monolithic architecture communication between application usually happen on localhost with application-defined port or open socket, In Microservices it's DevOps responsibility to establish a communication between Services and there responsiveness to keep the application alive.

The Microservice architecture diagram can go from simple to complex, the DevOps needs to have better planning what's the clear relationship between every component for example

- Nginx(80) depends on Tomcat (8080) for upstream server
- Tomcat (8080) depends on Redis (6349) to store the data on the cache.
- Nginx (80) has no relationship with Redis (6349)

By adopting Microservice architecture the DevOps is achieving the following

- Loosely coupled Container
- Each Container can easily unit tested
- Each Container can easily deployed
- Each Container can easily manage (Ask your self ??)
- Ability to have CI/CD in much-defined way
- Can Owned by small Team (Is it ?? What if you have more than 15+ containers) this is hypothetical stuff

When the DevOps achieve something on the first draft, it works like a charm and mostly you will get rewarded too now it's time to manage. Over the time the container may grow at this point the Development might become slow, as you dealing with more number of the configuration file and each container have different versions and different type of Environment like stage/production development with having different deployment strategies. This may become overhead, so designing Microservices might tempting but it comes with the cost also especially your dollar forecasts.

This problem can be solved by adapting the right pattern from the day-0 of your design and constantly reviewing when more container is getting added to Microservices cluster before it gets out of your hand.

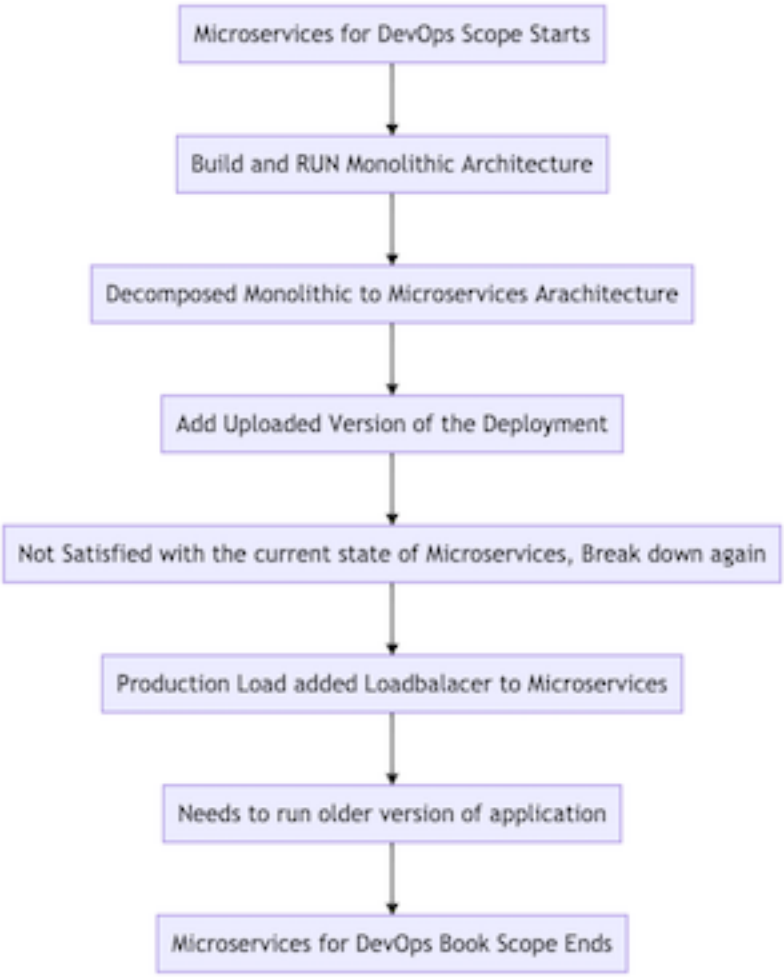
Well, that's enough of the lecture let's get to the practice, in the upcoming section we will learn what we going to build, then putting everything on the monolithic architecture, decomposing it into microservices architecture ad discuss various issues around this.

LAB Setup

4 MINUTE READ

Please pay close attention to LAB instruction, as this going to be building blocks for your knowledge.

Overall Microservices for DevOps will go through the below workflow



Microservices for DevOps Scope

Prerequisites

Before running this make sure you have installed **Docker** or **Podman** in your preferred operating system. Once you have **Docker** or **Podman** installed make sure you have cloned the required

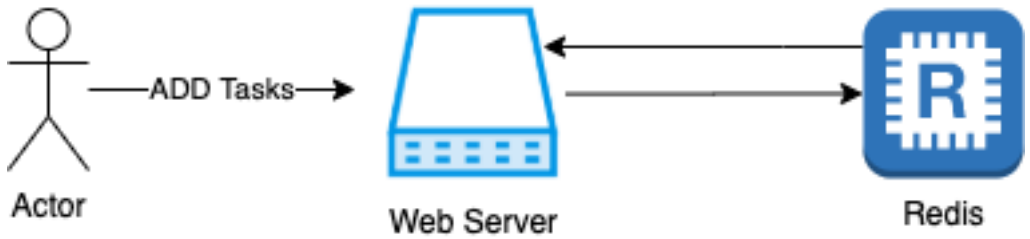
repository for LAB Instruction

```
$ git clone https://github.com/anishnath/Microservices-For-DevOps.git
```

MyNotes Demo Application

MyNotes application is a simple demo web application which takes one data as user input and store it into Redis Cache and display it to the end-user. Throughout this book we will cover the different state of microservices that MyNotes web application will go.

Here is the high-level flow diagram of the MyTask web application



Application Flow if MyTasks Web Application

Note: Once you understand the concept, you can overlay this Model to your Docker image.

Docker Images

These are the Docker pre-built docker images we are going to use for Microservices Demo, alternatively, you can use your docker Image by following the concepts explained in the upcoming chapter

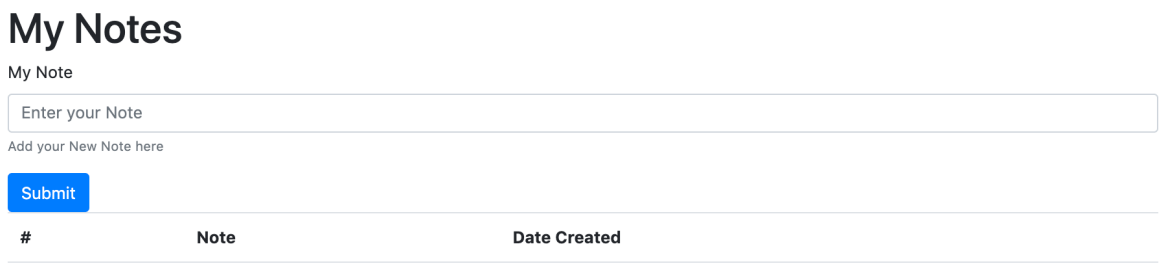
Docker Image	Description
redis	This Image contains Redis server
anishnath/demo:monolithic	This Image contains redis and Tomcat clubbed it to one Monololithic image
anishnath/demo:v1	Application running on Tomcat V1
anishnath/demo:v2	Application running on Tomcat V2
anishnath/demo:v3	Application running on Tomcat V3
anishnath/demo:api	API Server connecting to REDIS
anishnath/demo:nginx	Nginx deployment connecting to upstream server
anishnath/demo:nginx-elb	Nginx deployment connecting to two upstream server

- **Redis**: **MyTask** application uses Redis to store the myTask Data, here is the way you can pull the Redis image and start the Redis server on port 6349

```
$ docker pull redis
```

```
$ docker run -p 6349:6349 redis  redis-server --port 6349
1:C 21 Mar 2020 15:56:49.694 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 21 Mar 2020 15:56:49.694 # Redis version=5.0.8, bits=64, commit=00000000, modifi\
ed=0, pid=1, just started
1:C 21 Mar 2020 15:56:49.694 # Configuration loaded
1:M 21 Mar 2020 15:56:49.696 * Running mode=standalone, port=6349.
1:M 21 Mar 2020 15:56:49.696 # WARNING: The TCP backlog setting of 511 cannot be enf\
orced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
1:M 21 Mar 2020 15:56:49.696 # Server initialized
1:M 21 Mar 2020 15:56:49.696 # WARNING you have Transparent Huge Pages (THP) support\
 enabled in your kernel. This will create latency and memory usage issues with Redis\
. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepag\
e/enabled' as root, and add it to your /etc/rc.local in order to retain the setting \
after a reboot. Redis must be restarted after THP is disabled.
1:M 21 Mar 2020 15:56:49.696 * Ready to accept connections
```

- **anishnath/demo:v1**: The **v1** of **MyTask** application is very straight forward, just add your note in the Redis Cache Server and display it to end-user



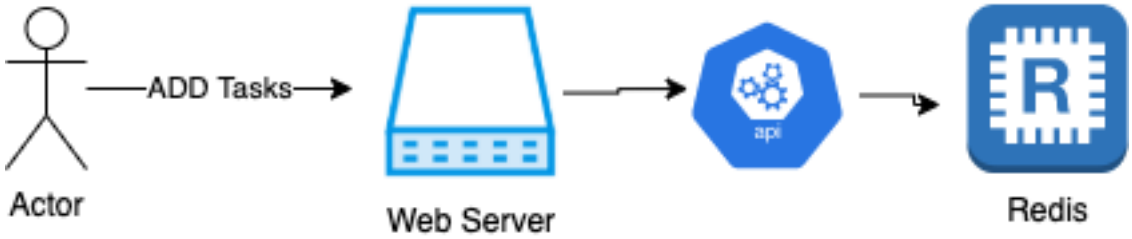
anishnath/demo:v1

```
$docker pull anishnath/demo:v1
$docker run -p 8080:8080 anishnath/demo:v1
```

- **anishnath/demo:v2**: The v2 of MyNote application is having an expiry option of each note as shown in the below diagram

anishnath/demo:v2

- **anishnath/demo:v3**: The v3 of MyNote application is having the architecture changes. There is API layer added which means the web application is calling API service to store and retrieve data in Redis Cache, There is no UI or logic changes



Added API Layer

- **anishnath/demo:v4** : This v4 vsersion is Just UI Improvement
- **anishnath/demo:nginx**: A Nginx reverse proxy serving one upstream server.

```
upstream dynamic {
        server demo:8080;
}
```

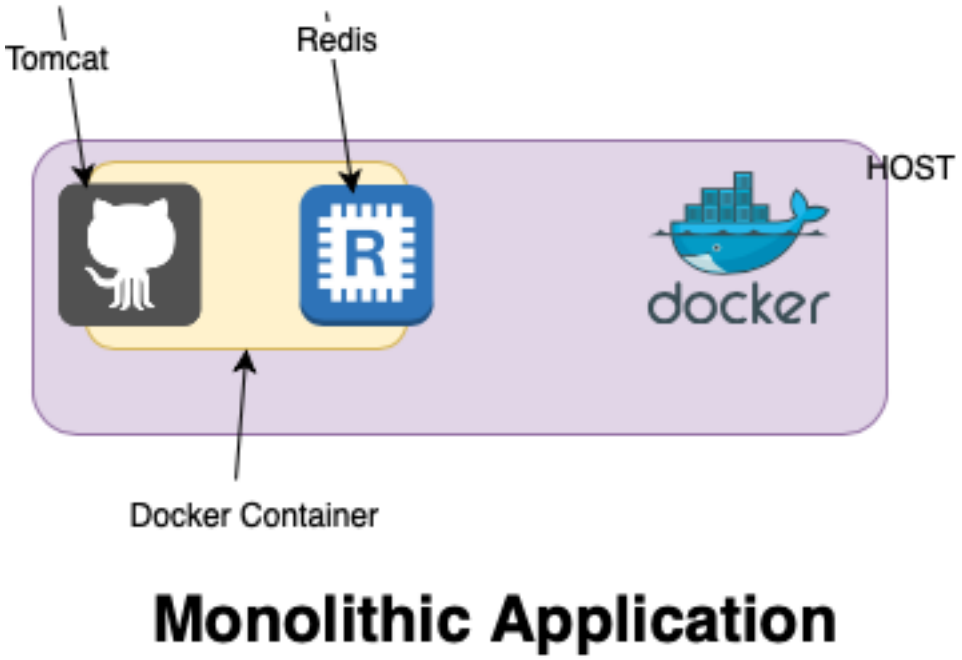
- **anishnath/demo:nginx-elb**: A Nginx reverse proxy serving two upstream server, here is the snippet of the configuration file

```
upstream dynamic {
        server demo:8080;
        server demo1:8080;
}
```

Monolith To MicroServices

6 MINUTE READ

Let's just start to decompose the **MyNotes** monolithic application. As shown in the below diagram MyNotes application is deployed in Tomcat and uses Redis to store data in-memory.



MyTask Monolithic Applciation

This single monolithic **MyNotes** application can be deployed using the `docker-compose` or `docker run` command. The Dockerfile[1] is part of the Github and container image `anishnath/demo:monolithic` used in this demo already pushed to docker hub

The `docker-compose-v0.yml` file

[1] https://raw.githubusercontent.com/anishnath/Microservices-For-DevOps/master/files/Docker.monolithic

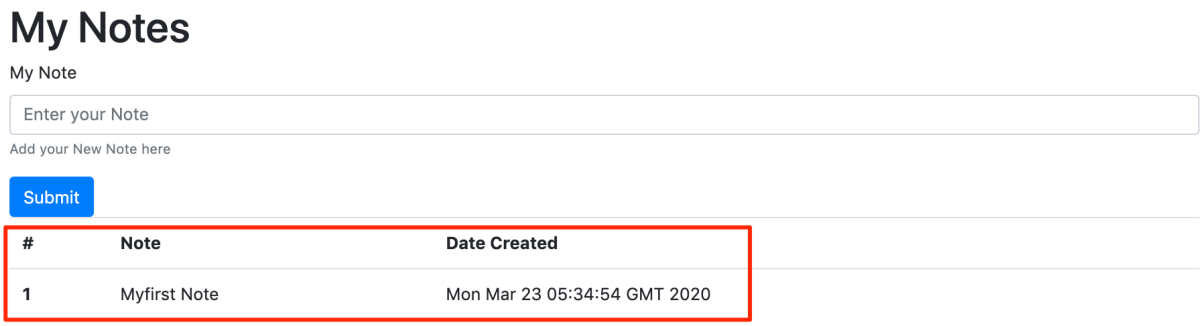
```
services:
demo:
container_name: demo
hostname: demo
image: anishnath/demo:monolithic
ports:
- 8080:8080
version: '3'
```

```
$ docker-compose -f docker-compose-v0.yml up -d
```

Verify the Service is up and running

```
$ docker-compose -f docker-compose-v0.yml  ps
Name Command State  Ports
-----------------------------------------------------------------
demo /data/monolithic.sh Up  6379/tcp, 0.0.0.0:8080->8080/tcp
```

and then browse the URL http://localhost:8080 to access the My Notes application, and add some dummy entry as shown below



Sample notes added

Decomposition of Monolith

At first learn your application, breakdown this to as minute details and try to establish the relationship between containers on service discoverability, our objective is to have ***one container one application***.

Application	Depends_on	Exposed PORT	Externaly Exposed	Container Security
Tomcat	Redis	8080	Y	N
Redis	-	6379	N	N

Take a Break

Before going from here DevOps folks, it good to invest some time on knowing the common pattern and antipattern drafted for Microservices architecture they are, many common and useful design, communication, and integration patterns.

The goal of this book not to covers the Microservices pattern and antipatterns, once you have gone with those details in internet or the book of your choice, it's easy to overlay the practical concept which you will gain from this book

Implementing Microservices is interesting and worth rewarding your skill as well managing your production load, but before that, you need to address all your application concerns in your Microservices architecture and how you're going to solve them like

- Config Management:
- Service Discovery
- Load Balancing
- API Gateway
- Security Concern
- Logging
- Metrics
- Resilience and fault tolerance
- Autoscaling and Self Healing
- Packaging, deployment, and scheduling
- Job management
- Singleton application

This book will try to address most of the concerns in the upcoming sections, lets' move on to the next phase.

Microservices Service Discovery

6 MINUTE READ

While breaking down the **MyNotes** application, we have come to one conclusion that the Monolithic application needs additional two container and the service discovery needs to happen from **Web->Redis**

- One Separate Container for Tomcat Web Server which **depends_on** Redis and expose externally.
- One Separate container for Redis

Application	Depends_on	Exposed PORT	Externaly Exposed	Container Security
Tomcat	Redis	8080	Y	N
Redis	-	6379	N	N

The **Service discovery**: maintain a list of service instances that are available for work within a microservice domain. This can be enabled with the use of `consul`, `DNS Discovery` like mechanism

Here is the docker-compose-v1.yml[2] which defines two services and the **Service discovery** is happening with **hostname** and **exposed port** and the communication is enabled with **depends_on** within docker which resolves the domain name.

```
services:
  redis-master:
    command:
    - redis-server
    - --port
    - '6379'
    container_name: redis-master
    hostname: redis-master
    image: redis
    ports:
    - 6379:6379

  demo:
    container_name: demo
    depends_on:
```

[2] https://raw.githubusercontent.com/anishnath/Microservices-For-DevOps/master/files/docker-compose-v1.yml

```
      - redis-master
    hostname: demo
    environment:
        - REDIS_SERVER=redis-master
    image: anishnath/demo:v1
    ports:
    - 8080:8080
version: '3'
```

Note: If you don't provide `depends_on` tag in docker-compose file, the conatiner with name demo will not able to contact redis via hostname redis-master and the service discovery will fail.

The DevOp's responsibility is to enable the service discovery is very critical and making it agnostic with DNS name. Imagine you have chosen a dynamic IP address to reach your service, this kind of design will lead to scaling issues as your container will scale up and down and DHCP will not guarantee you will have the same IP to reach your service which results in overall application to go down.

The LAB

In the previous exercise, we have set up a Monolithic MyNote application

```
$ docker-compose -f docker-compose-v0.yml ps
Name Command State  Ports
-----------------------------------------------------------------------
demo /data/monolithic.sh Up  6379/tcp, 0.0.0.0:8080->8080/tcp
```

Now we will override the Monolith application to a first draft of Microservices, run the below command

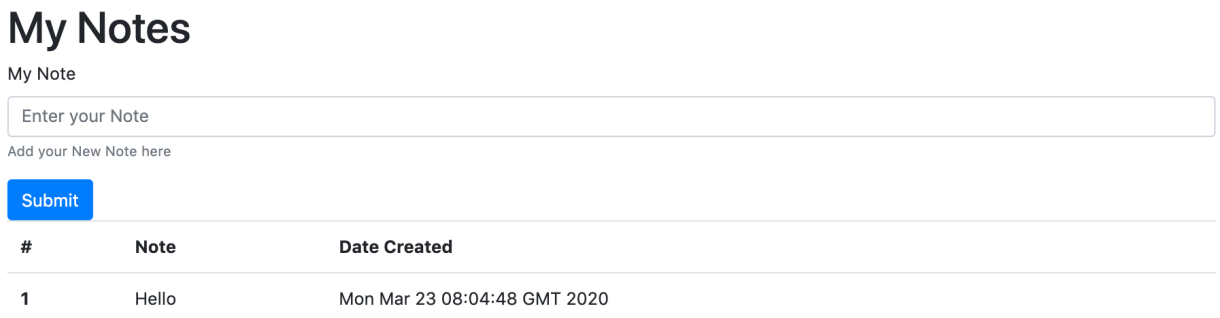
```
$ docker-compose -f docker-compose-v1.yml up -d
Starting redis-master ... done
Starting demo ... done
```

Check the process is up and running

```
$ docker-compose -f docker-compose-v1.yml ps
Name  Command State Ports
-------------------------------------------------------------------------------
demo /opt/tomcat/bin/catalina.s ... Up  0.0.0.0:8080->8080/tcp
redis-master docker-entrypoint.sh redis ... Up  0.0.0.0:6379->6379/tcp
```

The `docker-compose ps` confirms two services are running and there exposed port and command uses for there ENTRY POINTS

Now the application is up, browses the URL http://localhost:8080 to access the My Notes application, and add some dummy entry as shown below



Microservices first Draft

Cool: At least we can bring up new containers and establishing the link between then and overriding the existing system.

Security Concern

Did you spot the security Issue in above docker-compose file, one thing is obvious it has not configured with right container capabilities, insecure communication, and user privilege apart from that look at Redis service, The Redis port `6379` is exposed externally which has to prevent as exercised in above table.

```
ports:
    - 6379:6379
```

what about demo services, when we look at the env, the demo is connecting to Redis server without any passwords, and if this issue is fixed how the demo application with pass the password, ENV variable is not a good choice as docker inspect your reveal the password

This kind of CM issues needs to addressed on a timely basis and should be part of your Microservices Architectural design

```
environment:
        - REDIS_SERVER=redis-master
        - REDIS_PASSWORD="myexposedpassword in environment varaible"
```

Microservices Autoscaling

7 MINUTE READ

The **MyNotes** web application is decomposed in two containers and obtained the first draft of Microservices services, and we have a new version `anishnath/demo:v2` of this application as shown in the below diagram a new form field is added

New version

There is a new requirement to run this application with a certain number of replica (6) to keep up the production up and serve the demand.

The DevOps has compiled the docker-compose-v2.yml[3] to serve the need with **deploy** option

```
deploy:
        mode: replicated
        replicas: 6
```

The full compose file

```
services:
  redis-master:
    command:
    - redis-server
    - --port
    - '6379'
    container_name: redis-master
    hostname: redis-master
    image: redis
    ports:
```

[3] https://raw.githubusercontent.com/anishnath/Microservices-For-DevOps/master/files/docker-compose-v2.yml

```
    - 6379

  demo:
    container_name: demo
    depends_on:
      - redis-master
    hostname: demo
    deploy:
        mode: replicated
        replicas: 6
    environment:
        - REDIS_SERVER=redis-master
    image: anishnath/demo:v2
    ports:
    - 8080:8080
version: '3'
```

The LAB

```
$ docker-compose -f docker-compose-v2.yml up -d
WARNING: Some services (demo) use the 'deploy' key, which will be ignored. Compose d\
oes not support 'deploy' configuration - use `docker stack deploy` to deploy to a sw\
arm.
Recreating redis-master ... done
Recreating demo ... done
```

What did you just notice ??

Compose does not support 'deploy' configuration - use `docker stack deploy` to deploy to a swarm.

These kinds of issues occur when dealing with container-related technology like Docker, Podman, the `deploy` option is not supported in the `docker-compose up` instead we need to use `docker stack` as suggested, so from here onwards we will use `docker stack` command, in the true DevOps world learn to think to adapt fast.

Note: `docker stack` requires swarm mode, please initialize your swarm cluster

```
$ docker swarm init
```

and let's clean up the old deployment, not needed though we are just making our LAB clean

```
$ docker-compose -f docker-compose-v0.yml kill
$ docker-compose -f docker-compose-v1.yml kill
```

Create a new stack with stack name **mynotes**

```
$ docker stack deploy -c docker-compose-v2.yml  mynotes
Ignoring deprecated options:
container_name: Setting the container name is not supported.
Creating network mynotes_default
Creating service mynotes_redis-master
Creating service mynotes_demo
```

List out the stack deployment as you can see 6 replicas of your demo application is running

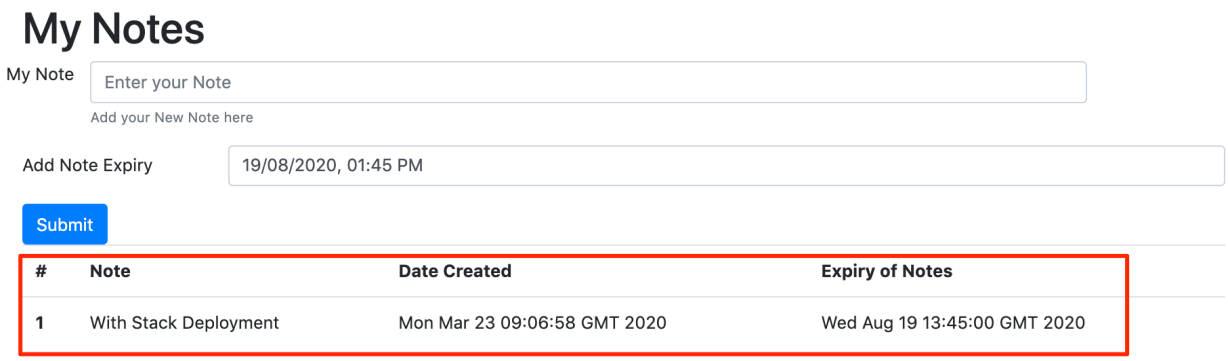
```
$ docker stack ps mynotes
ID  NAME IMAGE NODE  DESIRED STATE CURRENT STATE  ERROR PORTS
eut67jsr07z8  mynotes_demo.1 anishnath/demo:v2 docker-desktop  Running Running 34 se\
conds ago
ucab7t0xvran  mynotes_redis-master.1 redis:latest  docker-desktop  Running Running 4\
2 seconds ago
2nxit5zbpgi7  mynotes_demo.2 anishnath/demo:v2 docker-desktop  Running Running 34 se\
conds ago
z1k2kveekgjv  mynotes_demo.3 anishnath/demo:v2 docker-desktop  Running Running 34 se\
conds ago
p7b8jm04miq0  mynotes_demo.4 anishnath/demo:v2 docker-desktop  Running Running 34 se\
conds ago
r83nr3wy55w0  mynotes_demo.5 anishnath/demo:v2 docker-desktop  Running Running 34 se\
conds ago
dmn50jsk8ef1  mynotes_demo.6 anishnath/demo:v2 docker-desktop  Running Running 34 se\
conds ago
```

Once the stack is deployed the required services are automatically created as shown with their state

```
$ docker service ls
ID  NAME MODE  REPLICAS  IMAGE PORTS
2e8g5fnw1ekm  mynotes_demo replicated  6/6 anishnath/demo:v2 *:8080->8080/tcp
0bp4q5vp1wfq  mynotes_redis-master replicated  1/1 redis:latest  *:30002->6379/tcp
```

Extra Note: For Kubernetes user, this stack creation is equivalent to Kubernetes deployment and service is equivalent to Kubernetes service object. The swarm cluster is your Kubernetes nodes which joins to Kubernetes master node.

Now the application is up, browses the URL http://localhost:8080 to access the My Notes application, and add some dummy entry as shown below



Stack Autoscaling

This is the initial deployment, the DevOps might need to scale-up and scale-down on need basis on production.

Let's scale down to `replicas: 2`

```
$ docker stack deploy -c docker-compose-v2.yml  mynotes
Ignoring deprecated options:
container_name: Setting the container name is not supported.
Updating service mynotes_redis-master (id: 0bp4q5vp1wfq69qlsk38srf8p)
Updating service mynotes_demo (id: 2e8g5fnw1ekm37k7iq4uyxhi0)
```

or

First get the service name

```
$ docker service ls
ID  NAME MODE  REPLICAS  IMAGE PORTS
2e8g5fnw1ekm  mynotes_demo replicated  2/2 anishnath/demo:v2 *:8080->8080/tcp
0bp4q5vp1wfq  mynotes_redis-master replicated  1/1 redis:latest  *:30002->6379/tcp
```

Then scale up/down to required level.

```
$ docker service scale mynotes_demo=2
mynotes_demo scaled to 2
overall progress: 2 out of 2 tasks
1/2: running [==================================================>]
2/2: running [==================================================>]
verify: Service converged
```

After that check the mynotes stack process is running

```
$ docker stack ps mynotes
ID  NAME IMAGE NODE  DESIRED STATE CURRENT STATE  ERROR PORTS
eut67jsr07z8  mynotes_demo.1 anishnath/demo:v2 docker-desktop  Running Running 11 mi\
nutes ago
ucab7t0xvran  mynotes_redis-master.1 redis:latest  docker-desktop  Running Running 1\
2 minutes ago
2nxit5zbpgi7  mynotes_demo.2 anishnath/demo:v2 docker-desktop  Running Running 11 mi\
nutes ago
```

While scaling down you might be noticing the services are up and running in the background, this happens because of rolling update deployment strategy

The deployment strategy places an important role here like your Blue Ocean, recreate, ramped, A/B Testing, etc, we will cover some of the aspects of this on upcoming sections.

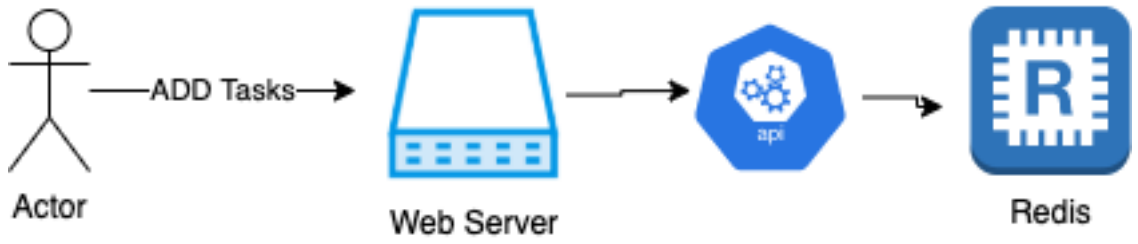
When it goes Complicated ?? : The microservices architecture should deal with the situation when scaling up or scaling down, The resources are limited even in the production system, the related external resources like CPU/Memory/Volume mapping are complicated, when scaling up you might encounter in a situation not all replicas are running and they are crashing, there condition of crashed should be documented and instead of applying a workaround, thing how they can be solved on Microservices deployment Architecture level which will server in long run and help in migration also, let's says you are moving from Docker to Kubernetes.

Microservices API gateway

8 MINUTE READ

In a Microservices architecture review which should happen regularly, it was found that the previous design is not using the modular concept, there is a need for API gateway.

The **MyNotes** web application should use an API to interact with REDIS and focus on just serving the web page. The API gateway should server all the required functionality



Added API Layer

So now we have three containers of MyNotes application and we have updated our working table with a new entry for API and established new service discoverability. The choice of exposing API externally is left to the end-user.

Application	Depends_on	Exposed PORT	Externaly Exposed	Container Security
Tomcat	API	8080	Y	N
API	Redis	8080	N	N
Redis	-	6379	N	N

DevOps has created a new image for the same `anishnath/demo:api` and while doing so a new version of the web application also comes in place `anishnath/demo:v3`

The DevOps has compiled the docker-compose-v3.yml[4] and added a new service with name **api** and updated the required service discoverability for each service.

[4] https://raw.githubusercontent.com/anishnath/Microservices-For-DevOps/master/files/docker-compose-v3.yml

```yaml
services:
  redis-master:
    command:
    - redis-server
    - --port
    - '6379'
    container_name: redis-master
    hostname: redis-master
    image: redis
    ports:
    - 6379

  demo:
    container_name: demo
    depends_on:
      - api
    hostname: demo
    deploy:
        mode: replicated
        replicas: 2
    environment:
      - REDIS_API_PORT=8080
      - REDIS_API_SERVER=api
    image: anishnath/demo:v3
    ports:
    - 8080:8080

  api:
    container_name: api
    depends_on:
      - redis-master
    hostname: api
    environment:
        - REDIS_SERVER=redis-master
        - REDIS_API_SERVER=api
        - REDIS_API_PORT=6379
    image: anishnath/demo:api
    ports:
    - 8080

version: '3'
```

The LAB

Now it's time to update the existing stack deployment to serve the new deployment

```
$ docker stack deploy -c docker-compose-v3.yml  mynotes
Ignoring deprecated options:
container_name: Setting the container name is not supported.
Creating service mynotes_api
Updating service mynotes_redis-master (id: 0bp4q5vp1wfq69qlsk38srf8p)
Updating service mynotes_demo (id: 2e8g5fnw1ekm37k7iq4uyxhi0)
```

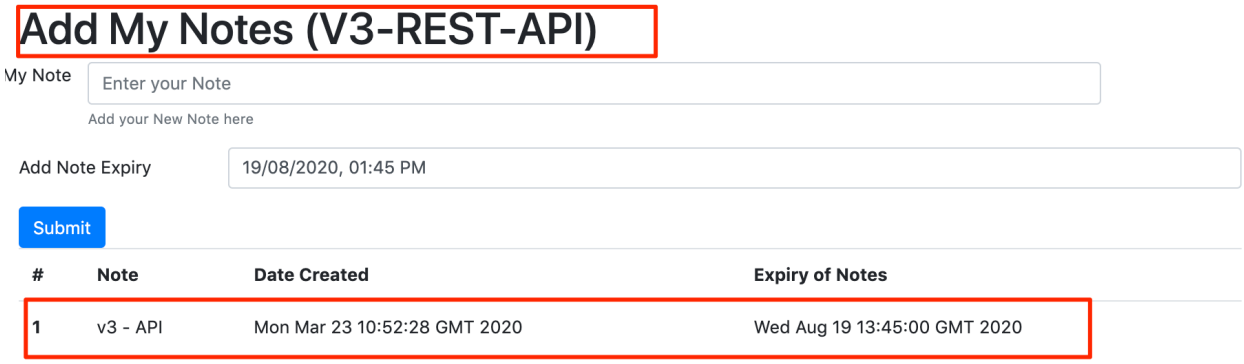
```
$ docker stack  ps mynotes
ID  NAME IMAGE  NODE  DESIRED STATE CURRENT STATE  ERROR PORTS
x76sxzu2aknq  mynotes_demo.1 anishnath/demo:v3  docker-desktop  Running Running 2 mi\
nutes ago
ohko6dolhz3y  mynotes_api.1  anishnath/demo:api docker-desktop  Running Running 3 mi\
nutes ago
eut67jsr07z8  mynotes_demo.1 anishnath/demo:v2  docker-desktop  Shutdown  Shutdown 2\
 minutes ago
ucab7t0xvran  mynotes_redis-master.1 redis:latest docker-desktop  Running Running 2 \
hours ago
k4ievnwa8805  mynotes_demo.2 anishnath/demo:v3  docker-desktop  Running Running 2 mi\
nutes ago
2nxit5zbpgi7 \_ mynotes_demo.2 anishnath/demo:v2  docker-desktop  Shutdown  Shutdown\
 2 minutes ago
```

While doing so, we can notice the older version `anishnath/demo:v2` is shut down and the new version `anishnath/demo:v3` is up and running in the stack.

and a new service being added to the stack

```
$ docker service ls
ID  NAME MODE  REPLICAS  IMAGE  PORTS
t6gpy9v6ldcp  mynotes_api  replicated  1/1 anishnath/demo:api *:30003->8080/tcp
2e8g5fnw1ekm  mynotes_demo replicated  2/2 anishnath/demo:v3  *:8080->8080/tcp
0bp4q5vp1wfq  mynotes_redis-master replicated  1/1 redis:latest *:30002->6379/tcp
```

The application is up, browse the URL http://localhost:8080 to access the My Notes application, and add some dummy entry as shown below



Service exposed using API gateway

With this we have just achieved the third version of Microservice architecture, just taking baby steps

The design of API gateway be should taken in consideration in the way, this service is agnostic of a client which means - It can easily scale up or down - Should act as a proxy for related services - Remain consistent across each REST API call via different containers or any external REST API Method - Should depend on readiness Probe and for liveness Probe

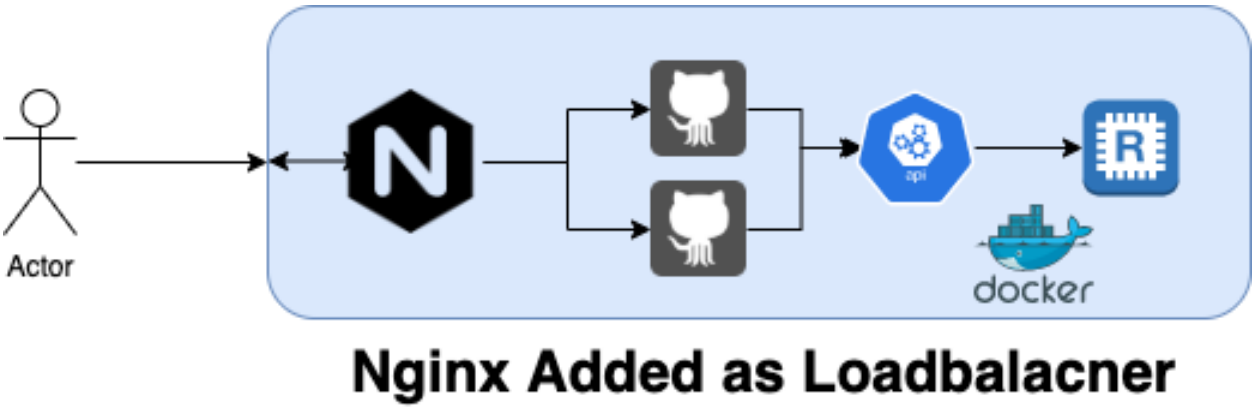
Security Concern

Every time we add a new service to our architecture we are creating one kind of service mesh, each new container should go with additional checks as they are different and their security needs are different

Microservices Loadbalancer

8 MINUTE READ

The existing version of Microservice is serving very well, all of sudden there is production load and the existing replication of the internal load balancer is not sufficient. In the next Microservices architecture team came up with the new design and chooses Nginx as a Reverse proxy which will keep the security issues and other needs as shown in the below diagram



Nginx as Reverese Proxy

At first we will update our service discoveriblity relation ship in the below ,

Application	Depends_on	Exposed PORT	Externaly Exposed	Container Security
Tomcat	API	8080	N	N
API	Redis	8080	N	N
Redis	-	6379	N	N
WEB	Tomcat/API/Redis	80	Y	N

DevOps has build a new image `anishnath/demo:nginx` for the use in the existing architecture.

DevOps has compiled the docker-compose-v4.yml[5] and added a new service definition name **web**

Note: The demo service (Tomcat) is no more exposing port **8080**

[5] https://raw.githubusercontent.com/anishnath/Microservices-For-DevOps/master/files/docker-compose-v4.yml

```yaml
services:
  redis-master:
    command:
    - redis-server
    - --port
    - '6379'
    container_name: redis-master
    hostname: redis-master
    image: redis
    ports:
    - 6379

  demo:
    container_name: demo
    depends_on:
      - api
    hostname: demo
    deploy:
        mode: replicated
        replicas: 2
    environment:
      - REDIS_API_PORT=8080
      - REDIS_API_SERVER=api
    image: anishnath/demo:v3
    ports:
    - 8080

  api:
    container_name: api
    depends_on:
      - redis-master
    hostname: api
    environment:
        - REDIS_SERVER=redis-master
        - REDIS_PORT=6379
    image: anishnath/demo:api
    ports:
    - 8080

  web:
    container_name: web
    depends_on:
      - demo
```

```
      - redis-master
      - api
    hostname: web
    image: anishnath/demo:nginx
    ports:
    - 80:80
version: '3'
```

The LAB

Let's update the existing stack deployment to serve Nginx as reverse proxy

```
$ docker stack deploy -c docker-compose-v4.yml  mynotes
Ignoring deprecated options:
container_name: Setting the container name is not supported.
Updating service mynotes_redis-master (id: 0bp4q5vp1wfq69qlsk38srf8p)
Updating service mynotes_demo (id: 2e8g5fnw1ekm37k7iq4uyxhi0)
Updating service mynotes_api (id: t6gpy9v6ldcpb4s4s6ettyj3y)
Creating service mynotes_web
```

Once the stack is updated

```
$ docker service ls
ID  NAME MODE  REPLICAS  IMAGE  PORTS
t6gpy9v6ldcp  mynotes_api  replicated  1/1 anishnath/demo:api *:30003->8080/tcp
2e8g5fnw1ekm  mynotes_demo replicated  2/2 anishnath/demo:v3  *:8080->8080/tcp
0bp4q5vp1wfq  mynotes_redis-master replicated  1/1 redis:latest *:30002->6379/tcp
huo36udqw8kg  mynotes_web  replicated  1/1 anishnath/demo:nginx *:80->80/tcp
```

The application is up, browse the URL http://localhost to access the My Notes application This is running on port 80, not 8080

Add My Notes (V4-REST-API)

My Note

Enter your Note

Add your New Note here

Add Note Expiry

19/08/2020, 01:45 PM

Submit

#	Note	Date Created	Expiry of Notes
1	Blank Note- 92	Mon Mar 23 11:42:13 GMT 2020	Wed Aug 19 13:45:00 GMT 2020
2	Blank Note- 41	Mon Mar 23 11:42:13 GMT 2020	Wed Aug 19 13:45:00 GMT 2020
3	Blank Note- 87	Mon Mar 23 11:42:12 GMT 2020	Wed Aug 19 13:45:00 GMT 2020
4	Blank Note- 77	Mon Mar 23 11:42:12 GMT 2020	Wed Aug 19 13:45:00 GMT 2020

Page serve through the Nginx Reverse Proxy

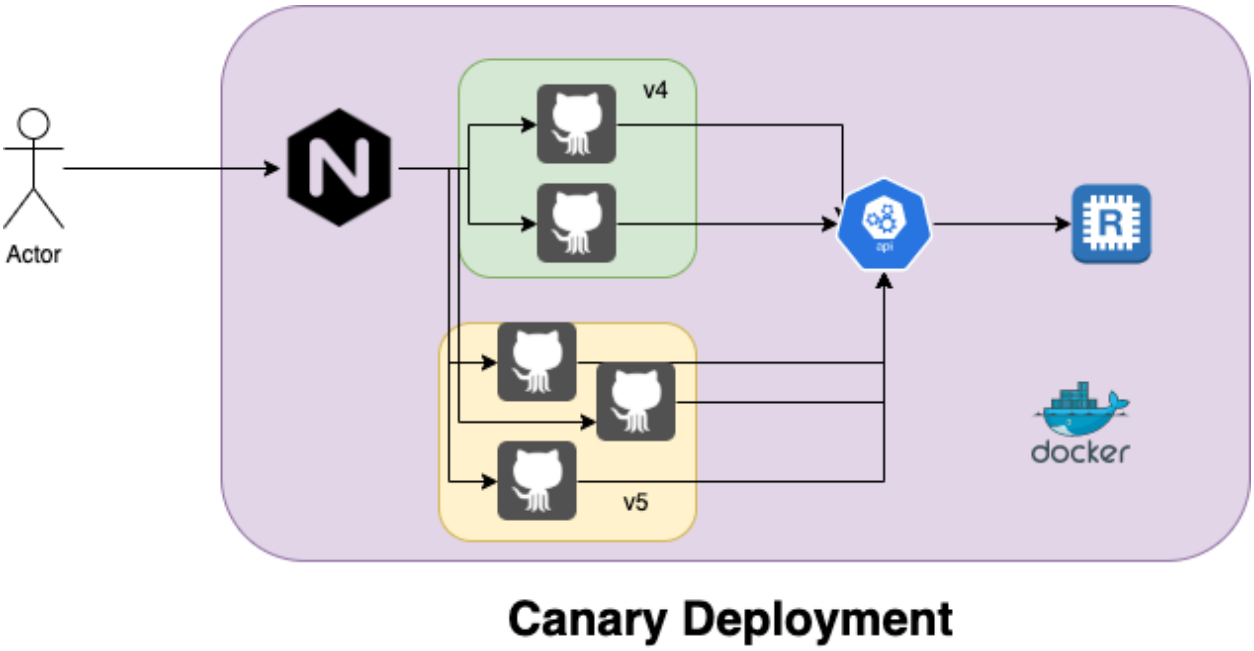
As you noticed from the monolithic application to Microservices decomposition, the Service Mesh is getting complex, though we are just adding one new service without touching our core application, like a pluggable component. That's the true beauty of using Microservices, but keep in mind, every new service will bring security risk and need to tackle independently.

Microservices Deployment Strategies

8 MINUTE READ

The production servers are running well with the load balancer setup in the previous chapter, now we have customers and they are happy, and the team has just released the new version to get in deployed in the stack.

This time they have decided to have two versions of application running in the production. A Microservices architecture meeting review conducted and come up **canary** deployment



Canary Deployment phase 1

- In Canary deployment, both version will be running on the production environment and serve the traffic
- let's run `anishnath/demo:v4` will run with 2 replicas.
- let's run `anishnath/demo:v5` will run with 4 replicas.
- `v4` will take out once the `v5` reported no issues

At first, we will update our service discoverability relationship in the below table as shown below a new service needs to be defined with name `demo1`

Application	Depends_on	Exposed PORT	Externaly Exposed	Container Security
demo(Tomcat)	API	8080	N	N
demo1(Tomcat)	API	8080	N	N
api	Redis	8080	N	N
Redis	-	6379	N	N
WEB(Nginx)	Tomcat/API/Redis	80	Y	N

DevOps has built a new image `anishnath/demo:v5` for the new version DevOps has built a new image `anishnath/demo:nginx-elb` for the Loadbalancer to support both upstream server **demo** and **demo1**, here is the snippet of `default.conf`

```
upstream dynamic {
        server demo:8080;
        server demo1:8080;
}
```

Once the images are ready, DevOps will apply this canary deployment

DevOps has compiled the docker-compose-v5.yml[6] to support the canary deployment and added one new service **demo1** and updated the **web** service to use latest config of Nginx

```
services:
  redis-master:
    command:
    - redis-server
    - --port
    - '6379'
    container_name: redis-master
    hostname: redis-master
    image: redis
    ports:
    - 6379

  demo:
    container_name: demo
    depends_on:
      - api
    hostname: demo
    deploy:
        mode: replicated
        replicas: 2
    environment:
```

[6] https://raw.githubusercontent.com/anishnath/Microservices-For-DevOps/master/files/docker-compose-v5.yml

```
      - REDIS_API_PORT=8080
      - REDIS_API_SERVER=api
    image: anishnath/demo:v4
    ports:
    - 8080

  demo1:
    container_name: demo1
    depends_on:
      - api
    hostname: demo1
    deploy:
        mode: replicated
        replicas: 4
    hostname: demo1
    environment:
      - REDIS_API_PORT=8080
      - REDIS_API_SERVER=api
    image: anishnath/demo:v5
    ports:
    - 8080

  api:
    container_name: api
    depends_on:
      - redis-master
    hostname: api
    environment:
        - REDIS_SERVER=redis-master
        - REDIS_PORT=6379
    image: anishnath/demo:api
    ports:
    - 8080

  web:
    container_name: web
    depends_on:
      - demo
      - demo1
      - redis-master
      - api
    hostname: web
    image: anishnath/demo:nginx-elb
```

```
    ports:
    - 80:80
version: '3'
```

The LAB

Let's begin with the canary deployment of the application, deploy the new stack

As we can see a new service is creating an updated required service to be mapped with the load balancer

```
$ docker stack deploy -c docker-compose-v5.yml mynotes
Ignoring deprecated options:
container_name: Setting the container name is not supported.
Updating service mynotes_redis-master (id: 0bp4q5vp1wfq69qlsk38srf8p
Updating service mynotes_demo (id: 2e8g5fnw1ekm37k7iq4uyxhi0)
Creating service mynotes_demo1
Updating service mynotes_api (id: t6gpy9v6ldcpb4s4s6ettyj3y)
Updating service mynotes_web (id: huo36udqw8kgbmy9ac43g5cmr)
image anishnath/demo:nginx-elb could not be accessed on a registry to record
its digest. Each node will access anishnath/demo:nginx-elb independently,
possibly leading to different nodes running different
versions of the image.
```

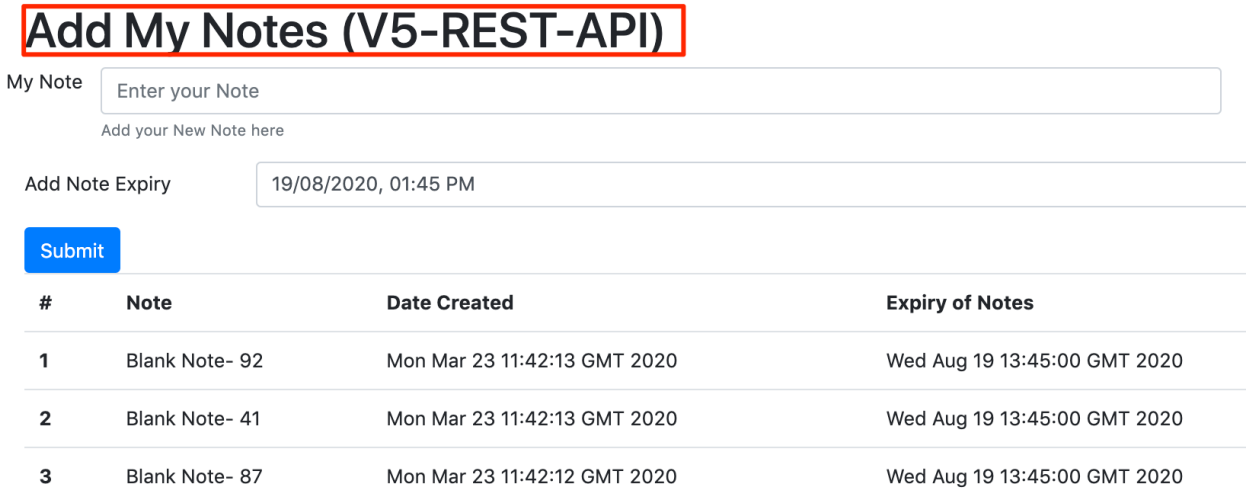
Check the services are running with the required replicas

```
$ docker service ls
ID  NAME MODE  REPLICAS  IMAGE  PORTS
t6gpy9v6ldcp  mynotes_api  replicated  1/1 anishnath/demo:api *:30003->8080/tcp
2e8g5fnw1ekm  mynotes_demo replicated  2/2 anishnath/demo:v4  *:8080->8080/tcp
kfx01yhw2dui  mynotes_demo1  replicated  4/4 anishnath/demo:v5  *:30004->8080/tcp
0bp4q5vp1wfq  mynotes_redis-master replicated  1/1 redis:latest *:30002->6379/tcp
huo36udqw8kg  mynotes_web  replicated  1/1 anishnath/demo:nginx-elb *:80->80/tcp
```

The canary deployment is created, it time for testing browse the URL http://localhost to access the MyNotes application and keep refreshing the page

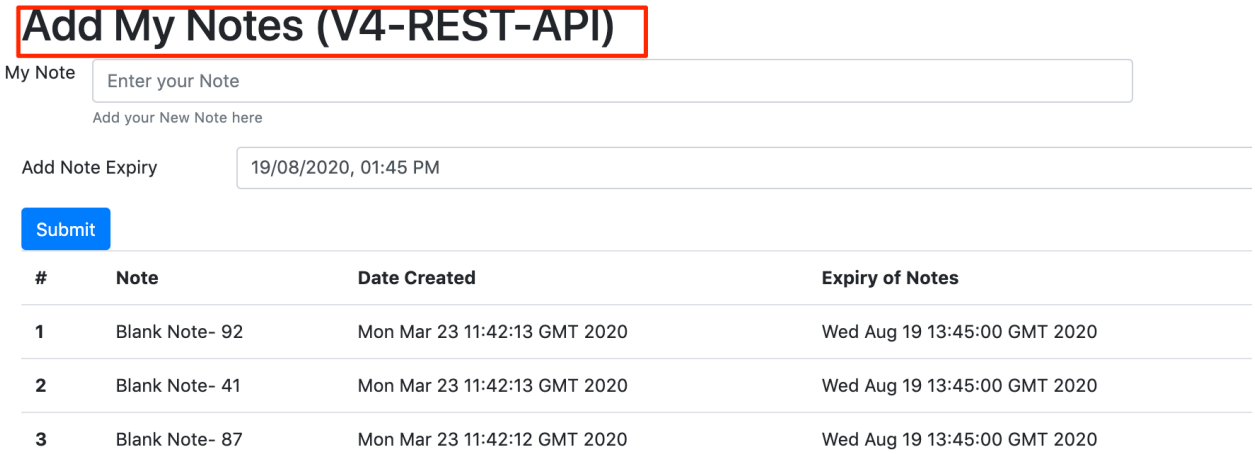
Every refresh will serve a different version because of the default round-robin policy of Nginx,

First refresh is shown `v5`



v5 Version

Another refresh has shown `v4` version



v4 Version

Tha's cool, now I know how to install multiple version of deployment, and the DevOps has got the confirmation that `v5` version is working as expected now it's time to take out `v4` version

```
$ docker service scale mynotes_demo=0
mynotes_demo scaled to 0
overall progress: 0 out of 0 tasks
verify: Service converged
```

`v4` is scaled down to `0`

```
[redacted]:files aninath$ docker service ls
ID                  NAME                   MODE         REPLICAS   IMAGE                      PORTS
t6gpy9v6ldcp        mynotes_api            replicated   1/1        anishnath/demo:api         *:30003->8080/tcp
2e8g5fnw1ekm        mynotes_demo           replicated   0/0        anishnath/demo:v4          *:8080->8080/tcp
kfx01yhw2dui        mynotes_demo1          replicated   4/4        anishnath/demo:v5          *:30004->8080/tcp
0bp4q5vp1wfq        mynotes_redis-master   replicated   1/1        redis:latest               *:30002->6379/tcp
huo36udqw8kg        mynotes_web            replicated   1/1        anishnath/demo:nginx-elb   *:80->80/tcp
```

v4 Version is taken out

Once the `v4` is scaled down to `0`, and later it will be deleted and the purpose of the canary version is completed.

By any means, if DevOps found that `v5` is not working as expected they can take that deployment and work on the next stable version.

Types of Deployment Strategy

Canary is not the only deployment strategy DevOps can apply, it depends based on business needs other kinds of deployment strategy DevOps can look for

- **Blue/Green**: Most common deployment strategy The new version (the blue version) is brought up in production but no traffic yet, while the users still use the stable version (the green version). When ready, the users are switched to the blue version. If a problem arises, you can switch back to the green version.
- **Rolling strategy for Canary deployment**: This is what we have gone with the lab exercise
- **A/B Deployment**: This kind of strategy lets you try a new version of the application in a limited way in the production environment. The DevOps user will define the new route for the API endpoints with a certain weight and another parameter
- **Custom Strategy**: Completely based on user behavior

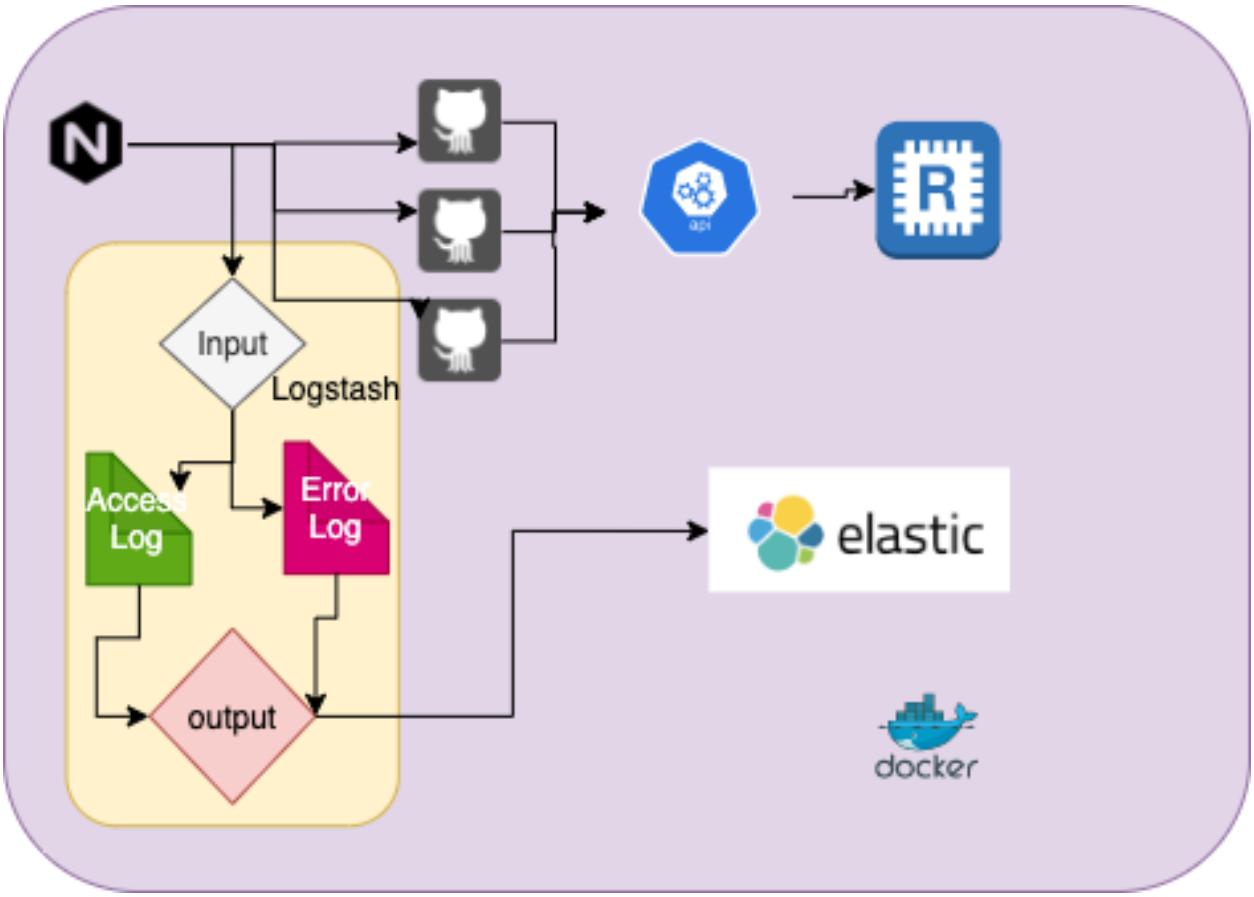
Hopefully, this gives you a good insight into what to do with the deployment strategies, this plays a crucial role way to change or upgrade an application, with only one objective how to avoid downtime and how to rollback when things doesn't go as per the planned.

Microservices Centralized logging

9 MINUTE READ

For a compliance perspective, there is a need to put all Nginx logs to a centralized Syslog server. Ok that's fine, let's have Microservices architecture review meeting and while conducting this meeting this solution came up

SPOILER: An imperfect solution ?? you will notice why



ELK deployed to Application

based on the above diagram first, we will update our service discoverability relationship in the below table

Application	Depends_on	Exposed PORT	Externaly Exposed	Container Security
demo(Tomcat)	API	8080	N	N
demo1(Tomcat)	API	8080	N	N
api	Redis	8080	N	N
Redis	-	6379	N	N
WEB(Nginx)	Tomcat/API/Redis	80 logstatsh	Y	N
logstash	elasticsearch	5140	N	N
elastic	-	9200	Y	N

The Nginx **access_log** and **error_log** will feed to Logstatsh and logstatsh will transfer this Information to elastic search.

This piece of Information demands the changes in Nginx docker image, but what kind of changes, do we have built-in support for Syslog server from Nginx, google it.

Google hit the result, The error_log[7] and access_log[8] directives support logging to Syslog. The following parameters configure logging to Syslog:

```
# Custom log format that also includes the host that processed the request
log_format logstash '$remote_addr - $remote_user [$time_local] "$host" '
                    '"$request" $status $body_bytes_sent '
                    '"$http_referer" "$http_user_agent"';

# Send logs to Logstash
access_log syslog:server=logstash:5140,tag=nginx_access logstash;
error_log syslog:server=logstash:5140,tag=nginx_error notice;
```

Let's have this conf file update in our new version of Nginx Image `anishnath/demo:nginxv1`

The `Dockerfile`

```
FROM nginx
COPY default.syslog.conf /etc/nginx/conf.d/default.conf
```

```
docker build -t anishnath/demo:nginxv1 -f Docker.nginx.syslog .
```

We have solved one piece of the puzzle, the next piece is configuring **logstatsh** like - Build logstash Docker image - Installed required plugin into it to support Syslog processing

The Docker image which does the Job is `anishnath/demo:logstash` and here is `Dockerfile` for the same

[7] http://nginx.org/en/docs/ngx_core_module.html#error_log
[8] http://nginx.org/en/docs/http/ngx_http_log_module.html#access_log

```
FROM logstash:5.5-alpine
ENV PLUGIN_BIN "/usr/share/logstash/bin/logstash-plugin"
RUN "$PLUGIN_BIN" install logstash-filter-grok
RUN "$PLUGIN_BIN" install logstash-input-syslog
RUN "$PLUGIN_BIN" install logstash-filter-date
RUN "$PLUGIN_BIN" install logstash-filter-useragent
RUN "$PLUGIN_BIN" install logstash-output-elasticsearch
COPY ./conf /etc/logstash
CMD ["-f", "/etc/logstash/logstash.conf"]
```

here is the new service definition for the **logstash**

```
 logstash:
    container_name: logstash
    depends_on:
      - elasticsearch
    hostname: logstash
    image: anishnath/demo:logstash
    ports:
    - 5140
```

The third piece is getting the elasticsearch which would be straight forward, and I have done no customization to the docker image.

```
 elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.5.2
    hostname: elasticsearch
    container_name: elasticsearch
    environment:
      - discovery.type=single-node
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    ports:
      - 9200:9200
      - 9300:9300
```

The final piece is to have a Kibana dashboard if you need to access UI component

```
kibana:
    container_name: kibana
    hostname: kibana
    links:
      - elasticsearch
      - logstash
    image: docker.elastic.co/kibana/kibana:7.5.1
    environment:
      - SERVER_HOST=0.0.0.0
      - ELASTICSEARCH_URL=http://elasticsearch:9200
    ports:
      - 5601:5601
```

The LAB

DevOps has compiled the `docker-compose-v6.yml` as per defined in the architecture.

Now deploy the **mynotes** stack

```
$ docker stack deploy -c docker-compose-v6.yml mynotes
Ignoring unsupported options: links, ulimits
Ignoring deprecated options:
container_name: Setting the container name is not supported.
Creating network mynotes_default
Creating service mynotes_demo
Creating service mynotes_demo1
Creating service mynotes_api
Creating service mynotes_web
Creating service mynotes_elasticsearch
Creating service mynotes_logstash
Creating service mynotes_kibana
Creating service mynotes_redis-master
```

Verify the **mynotes** services are up and running

```
$ docker service ls
ID  NAME  MODE  REPLICAS  IMAGE PORTS
ru59k5kuvebi  mynotes_api replicated  1/1 anishnath/demo:api  *:30007->8080/tcp
kcinb78h62cg  mynotes_demo  replicated  1/1 anishnath/demo:v4 *:30005->8080/tcp
pcifk30dxy1l  mynotes_demo1 replicated  1/1 anishnath/demo:v5 *:30006->8080/tcp
87p14tnca20v  mynotes_elasticsearch replicated  1/1 docker.elastic.co/elasticsearch/\
elasticsearch:7.5.2 *:9200->9200/tcp, *:9300->9300/tcp
obcbpqymyqwi  mynotes_kibana  replicated  1/1 docker.elastic.co/kibana/kibana:7.5.1 \
*:5601->5601/tcp
qjd32cbug35w  mynotes_logstash  replicated  1/1 anishnath/demo:logstash *:30008->514\
0/tcp
hq5ni7wjsy9x  mynotes_redis-master  replicated  1/1 redis:latest  *:30009->6379/tcp
l3tio8b5clzx  mynotes_web replicated  1/1 anishnath/demo:nginxv1  *:80->80/tcp
```

After all, services are ready, we can open up `http://localhost` in our web browser and work on **MyNotes** application

After making that request, we can look inside `ElasticSearch` to make sure there's log data saved by hitting the below endpoint

```
$ curl  http://localhost:9200/logstash-*/_search/?size=10&pretty=1
```

The result which shows nginx is pushing log data to logstash

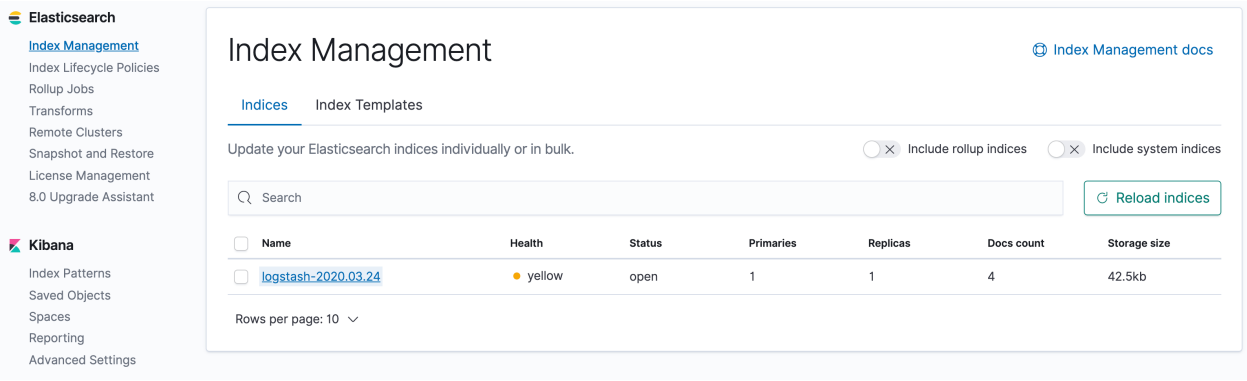
```
{
  "took": 20,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 5,
      "relation": "eq"
    },
    "max_score": 1,
    "hits": [
      {
        "_index": "logstash-2020.03.24",
        "_type": "nginx_access",
```

```
        "_id": "exBjDHEBm7_z2rr-O92Z",
        "_score": 1,
        "_source": {
          "severity": 6,
....
....

    ]
  }
}
```

The Nginx access_log is getting a push to elasticsearch, this can be visualized using the KIbana dashboard

Access Kibana dashboard by visiting http://0.0.0.0:5600 and configure new Index pattern with name `logstatsh-*`



Kibana Dashbaord

Once the index pattern is been configured, you are ready to create new visualization as defined by Kibana.

Security Consideration

We have added 3 new services to the service mesh, DevOps has introduced a new level of security risk. Assessing those risk and applying the remediation should be one of the primary tasks of DevOps

Final Note

So far, we have addressed much concern while dealing with microservices, it's not enough there are many points you need to take care when deploying anything on the production

- Did we added health check to container

- Is the container is followed the least privilege principle
- Is the image pushed is signed
- Do we say all logs are captured to a centralized server
- Do we have enough metrics to detect any failure to the system
- Do we have monitoring for application as well for your running cluster
- Do we have triggering mechanism
- Do we generate necessary events from the metrics or logs
- Do we have proper CI/CD in place to support the required deployment strategy
- Do we have the necessary way to update the system
- Do we have a way to audit the required configuration files

Though many of the question is not answered in this book, my apology for that but if you feel you can connect with me, please connect to my twitter handle @anish2good

Once again thanks for reading, appreciated your time

About Me

My Name is Anish Nath, I work as Security & Cloud Engineer for a Product company having 13+ year of IT experience, Some times I do Perform ethical hacking through Official HackerOne & Bugcrowd Channel,

The books that I write are a mechanism to support my own learning, I'm totally in awe of the Open Source community that has made this type of work possible.

Author of the Book

1. ***The Modern Cryptography Cookbook*** *(Leanpub[9])*
2. **Go Lang Cryptography for Developers** (Leanpub[10])
3. ***Python Cryptography*** *(Leanpub[11])*
4. ***Cryptography for Javascript Developer*** *(Leanpub[12])*
5. **Cryptography for PHP Developer** (Leanpub[13])
6. **Hello Dockerfile** (Leanpub[14])
7. **Kubernetes for DevOps** (Leanpub[15])
8. **Microservices for DevOpsc** (Leanpub[16])
9. **podman buildah skopeo runc** (Leanpub[17])
10. **Packet Analysis with Wireshark**. (Packtpub),

Visit my Latest Crypto Work @ https://8gwifi.org[18]

You can reach me through my twitter handle @anish2good[19]

[9] http://leanpub.com/crypto
[10] https://leanpub.com/cryptog
[11] https://leanpub.com/dockerfile
[12] http://leanpub.com/cryptojs
[13] https://leanpub.com/cryptophp
[14] https://leanpub.com/dockerfile
[15] https://leanpub.com/kube
[16] https://leanpub.com/microops
[17] https://leanpub.com/podman
[18] https://8gwifi.org
[19] https://twitter.com/anish2good