Overlay Network:

In the context of Docker, an overlay network is a network that spans multiple Docker daemon hosts, allowing containers on different hosts to communicate with each other. This becomes particularly useful in a distributed application where containers might be running on different machines but need to work together.

Let's say you are working on Kubernetes cluster (k8s) or Docker Swarm where in your whole network you have multiple machines. Inside the machines there are multiple containers are running. All together they are on a cluster.

In sort, if you use multiple hosts you need overlay network.

Let's do some practical on this. In order to do the project/practical I will user Docker Swarm.

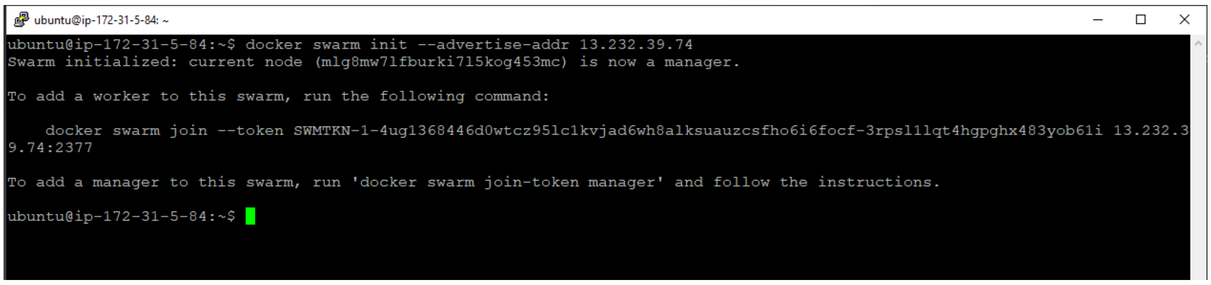
<u>What is Docker Swarm?</u>

Docker Swarm is a native clustering and orchestration solution for Docker. It allows you to create and manage a swarm of Docker nodes, turning them into a single, virtual Docker engine. This enables you to deploy and manage containers at scale, providing features for high availability, load balancing, and easy scaling of applications.

We will have separate project on Docker Swarm, for now just follow as per below.

- Create 2 ec2 instances. One will be our master another one will be worker.
- In one node which you want to make master, run docker swarm init command with below arguments

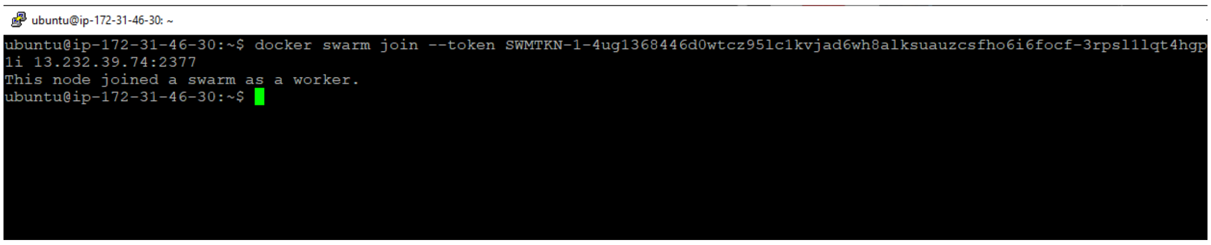
$docker swarm init --advertise-addr 13.232.39.74



```
ubuntu@ip-172-31-5-84:~$ docker swarm init --advertise-addr 13.232.39.74
Swarm initialized: current node (mlg8mw7lfburki715kog453mc) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-4ug1368446d0wtcz95lc1kvjad6wh8alksuauzcsfho6i6focf-3rpsl1lqt4hgpghx483yob61i 13.232.3
9.74:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

ubuntu@ip-172-31-5-84:~$
```

Here in the command the ip is the public IP of you node/master which you want to make.

- Once you run that command you will get a token generated in your screen along with command , you need to run it to another node which will be your worker.

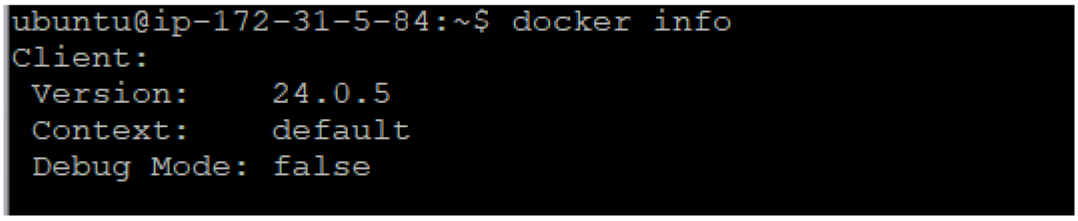


ubuntu@ip-172-31-46-30: ~

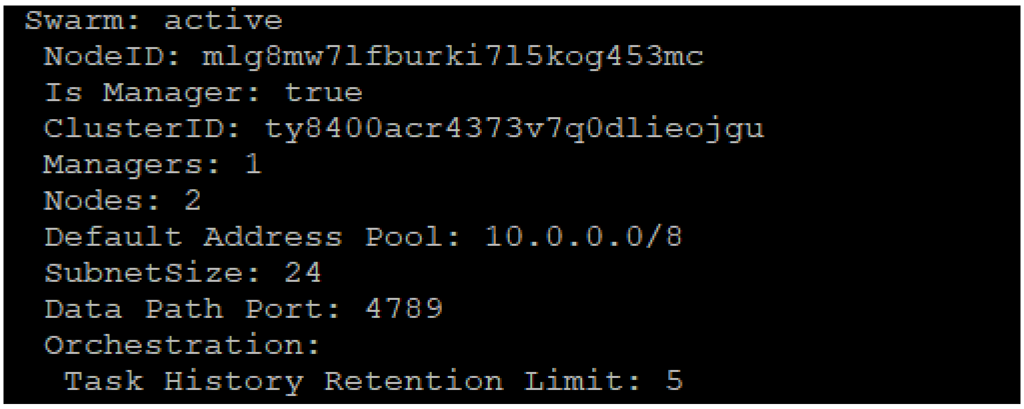
```
ubuntu@ip-172-31-46-30:~$ docker swarm join --token SWMTKN-1-4ug1368446d0wtcz95lc1kvjad6wh8alksuauzcsfho6i6focf-3rpsl1lqt4hgp
1i 13.232.39.74:2377
This node joined a swarm as a worker.
ubuntu@ip-172-31-46-30:~$
```

Now your manger and worker are connected to each other through docker swarm.

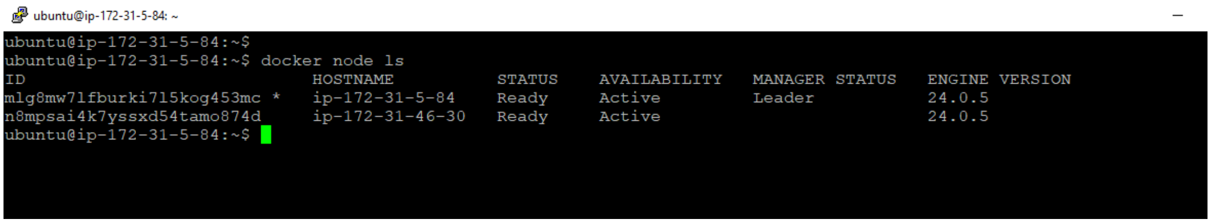
- Let's see more on this:-



```
ubuntu@ip-172-31-5-84:~$ docker info
Client:
 Version:    24.0.5
 Context:    default
 Debug Mode: false
```



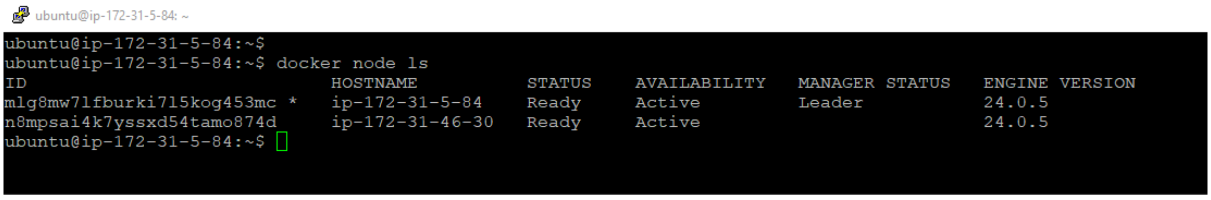
```
 Swarm: active
  NodeID: mlg8mw7lfburki715kog453mc
  Is Manager: true
  ClusterID: ty8400acr4373v7q0dlieojgu
  Managers: 1
  Nodes: 2
  Default Address Pool: 10.0.0.0/8
  SubnetSize: 24
  Data Path Port: 4789
  Orchestration:
   Task History Retention Limit: 5
```



ubuntu@ip-172-31-5-84: ~

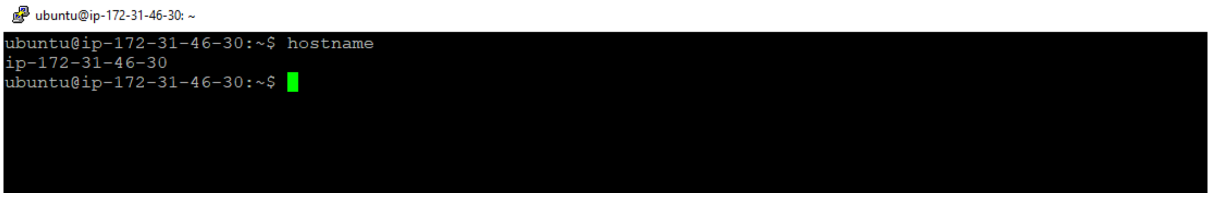
```
ubuntu@ip-172-31-5-84:~$
ubuntu@ip-172-31-5-84:~$ docker node ls
ID                            HOSTNAME           STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
mlg8mw7lfburki715kog453mc *   ip-172-31-5-84     Ready     Active         Leader           24.0.5
n8mpsai4k7yssxd54tamo874d     ip-172-31-46-30    Ready     Active                          24.0.5
ubuntu@ip-172-31-5-84:~$
```

In below screen shot you can see node details :-



ubuntu@ip-172-31-5-84: ~

```
ubuntu@ip-172-31-5-84:~$
ubuntu@ip-172-31-5-84:~$ docker node ls
ID                            HOSTNAME           STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
mlg8mw7lfburki715kog453mc *   ip-172-31-5-84     Ready     Active         Leader           24.0.5
n8mpsai4k7yssxd54tamo874d     ip-172-31-46-30    Ready     Active                          24.0.5
ubuntu@ip-172-31-5-84:~$
```



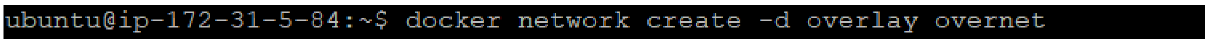
ubuntu@ip-172-31-46-30: ~

```
ubuntu@ip-172-31-46-30:~$ hostname
ip-172-31-46-30
ubuntu@ip-172-31-46-30:~$
```

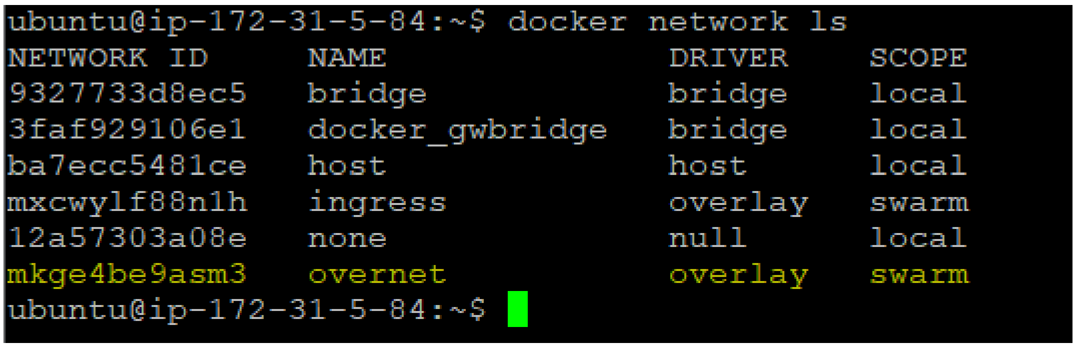
- Now our main project on overlay network.
- Create an overlay network first with below command

$docker network create -d overlay overnet

(-d is driver , for us driver is overlay
Overnet is the name of our network we have given.)



```
ubuntu@ip-172-31-5-84:~$ docker network create -d overlay overnet
```



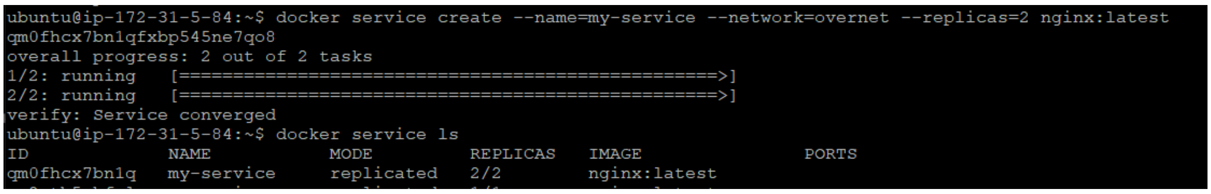
```
ubuntu@ip-172-31-5-84:~$ docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
9327733d8ec5   bridge            bridge    local
3faf929106e1   docker_gwbridge   bridge    local
ba7ecc5481ce   host              host      local
mxcwylf88n1h   ingress           overlay   swarm
12a57303a08e   none              null      local
mkge4be9asm3   overnet           overlay   swarm
ubuntu@ip-172-31-5-84:~$
```

- Now create a service with below arguments.

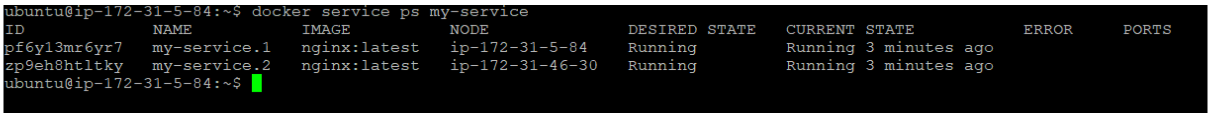
$docker service create --name=my-service --network=overnet --replicas=2 nginx:latest

[Service name is my-service, 2 replicas will be created using overnet network which we just created (overlay) and nginx containers will be running]

- The service is the main which will help both the node's containers to be connected through overlay network. It's kind of handshaking between multiple hosts containers.
- Below are the commands in screen shot to verify.



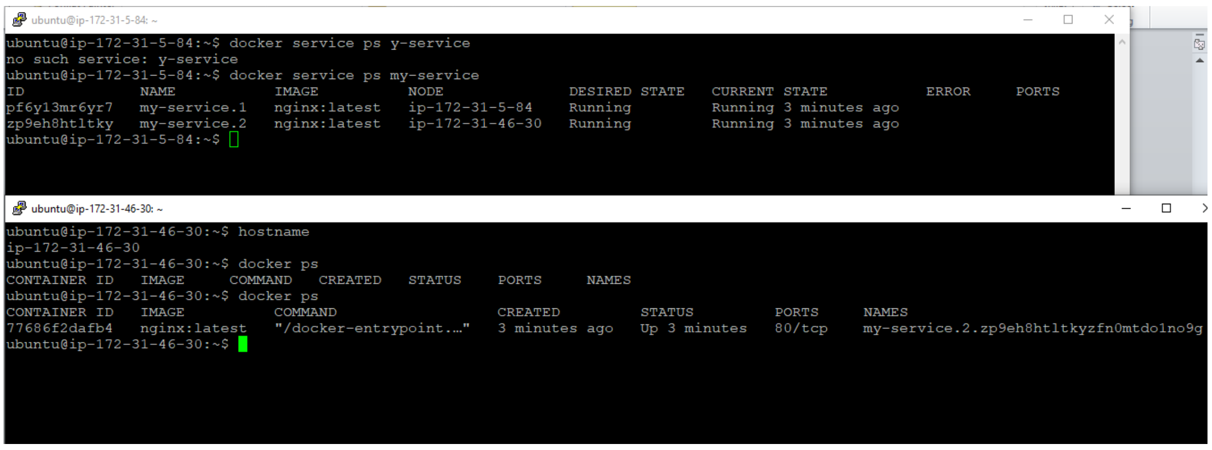
```
ubuntu@ip-172-31-5-84:~$ docker service create --name=my-service --network=overnet --replicas=2 nginx:latest
qm0fhcx7bn1qfxbp545ne7qo8
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
ubuntu@ip-172-31-5-84:~$ docker service ls
ID             NAME         MODE         REPLICAS   IMAGE          PORTS
qm0fhcx7bn1q   my-service   replicated   2/2        nginx:latest
```



```
ubuntu@ip-172-31-5-84:~$ docker service ps my-service
ID             NAME           IMAGE          NODE             DESIRED STATE   CURRENT STATE            ERROR     PORTS
pf6y13mr6yr7   my-service.1   nginx:latest   ip-172-31-5-84   Running         Running 3 minutes ago
zp9eh8htltky   my-service.2   nginx:latest   ip-172-31-46-30  Running         Running 3 minutes ago
ubuntu@ip-172-31-5-84:~$
```

$docker service ls

$docker service ps my-service

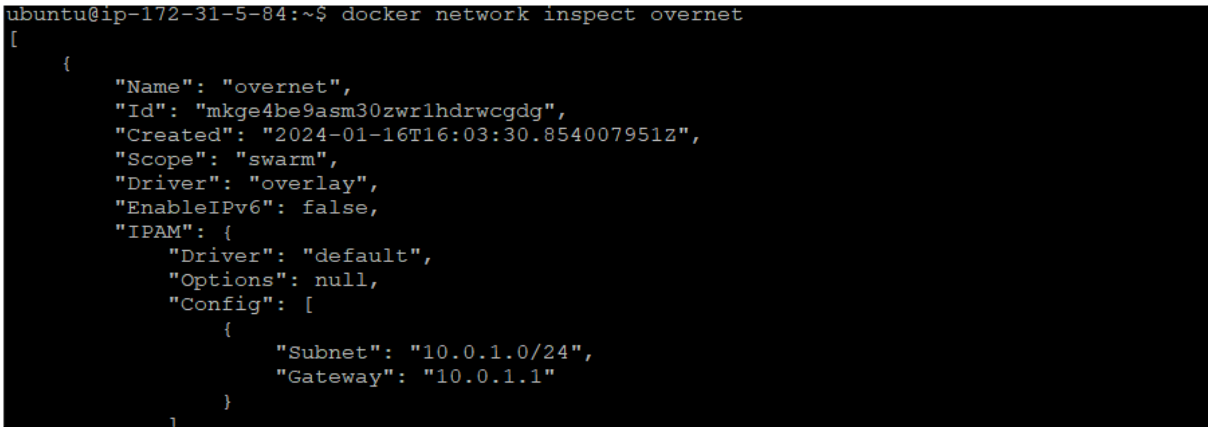


```
ubuntu@ip-172-31-5-84:~$ docker service ps y-service
no such service: y-service
ubuntu@ip-172-31-5-84:~$ docker service ps my-service
ID             NAME           IMAGE          NODE             DESIRED STATE   CURRENT STATE           ERROR     PORTS
pf6y13mr6yr7   my-service.1   nginx:latest   ip-172-31-5-84   Running         Running 3 minutes ago
zp9eh8htltky   my-service.2   nginx:latest   ip-172-31-46-30  Running         Running 3 minutes ago
ubuntu@ip-172-31-5-84:~$
```

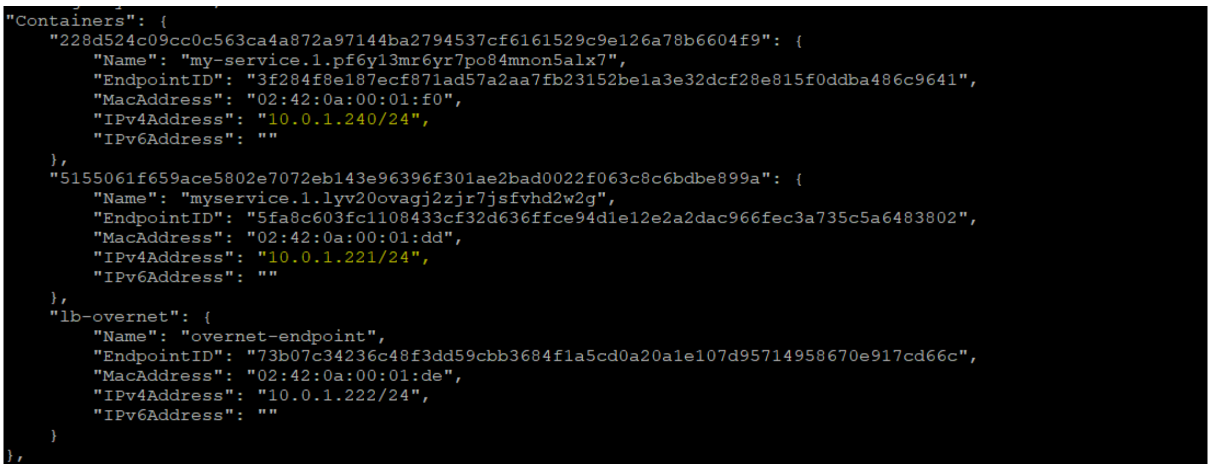
```
ubuntu@ip-172-31-46-30:~$ hostname
ip-172-31-46-30
ubuntu@ip-172-31-46-30:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu@ip-172-31-46-30:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
77686f2dafb4   nginx:latest   "/docker-entrypoint.…"   3 minutes ago   Up 3 minutes   80/tcp    my-service.2.zp9eh8htltkyzfn0mtdo1no9g
ubuntu@ip-172-31-46-30:~$
```

- Now inspect our overlay network and see if the containers are running on same subnet under overlay network or not.

  $ docker network inspect overnet



```
ubuntu@ip-172-31-5-84:~$ docker network inspect overnet
[
    {
        "Name": "overnet",
        "Id": "mkge4be9asm30zwr1hdrwcgdg",
        "Created": "2024-01-16T16:03:30.854007951Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.1.0/24",
                    "Gateway": "10.0.1.1"
                }
```



```
"Containers": {
    "228d524c09cc0c563ca4a872a97144ba2794537cf6161529c9e126a78b6604f9": {
        "Name": "my-service.1.pf6y13mr6yr7po84mnon5a1x7",
        "EndpointID": "3f284f8e187ecf871ad57a2aa7fb23152be1a3e32dcf28e815f0ddba486c9641",
        "MacAddress": "02:42:0a:00:01:f0",
        "IPv4Address": "10.0.1.240/24",
        "IPv6Address": ""
    },
    "5155061f659ace5802e7072eb143e96396f301ae2bad0022f063c8c6bdbe899a": {
        "Name": "myservice.1.lyv20ovagj2zjr7jsfvhd2w2g",
        "EndpointID": "5fa8c603fc1108433cf32d636ffce94d1e12e2a2dac966fec3a735c5a6483802",
        "MacAddress": "02:42:0a:00:01:dd",
        "IPv4Address": "10.0.1.221/24",
        "IPv6Address": ""
    },
    "lb-overnet": {
        "Name": "overnet-endpoint",
        "EndpointID": "73b07c34236c48f3dd59cbb3684f1a5cd0a20a1e107d95714958670e917cd66c",
        "MacAddress": "02:42:0a:00:01:de",
        "IPv4Address": "10.0.1.222/24",
        "IPv6Address": ""
    }
},
```

As you can see both containers are running in same subnet range in a same Docker network which is overlay. They both should be talk to each other.

Please do the practical with other image and try pinging. How to do that? Check my another post/document on Custom bridge network you will get the steps.

https://www.linkedin.com/posts/avik-dutta-ba4b2952_docker-host-custom-bridge-nw-activity-7151792517540147202-6bc-?utm_source=share&utm_medium=member_desktop

Here are some key points about Docker overlay networks:

Multi-Host Communication: Overlay networks enable communication between containers running on different Docker hosts. It essentially creates a virtual network that sits on top of the existing infrastructure, allowing containers to communicate seamlessly regardless of the physical machine they are on.

Encapsulation: When containers communicate over an overlay network, the data is encapsulated. This means that the details of how the communication happens are abstracted away, making it easier for containers to talk to each other without worrying about the intricacies of the underlying network.

Swarm Mode: Docker Swarm, which is Docker's native clustering and orchestration solution, often utilizes overlay networks. In a Swarm cluster, overlay networks help in connecting services and distributing them across multiple nodes while ensuring they can communicate effortlessly.

Service Discovery: Overlay networks facilitate service discovery. Containers can be referred to by their service names, and Docker's built-in DNS server takes care of routing requests to the appropriate container. This simplifies the way containers find and communicate with each other.

Security: Overlay networks provide isolation and security. Communication within the overlay network is confined to the containers connected to it, and the underlying physical network doesn't need to be aware of the details of container communication.

Dynamic Updates: Overlay networks support dynamic updates. If you add or remove containers from the network, the overlay network adapts to these changes automatically, making it flexible and scalable.

In summary, Docker overlay networks offer a powerful solution for networking in distributed and clustered environments. They allow containers to communicate seamlessly across multiple hosts, providing a virtual network that simplifies the complexities of distributed application architectures.