Accessing the Cluster: Understanding the Protocol

In this lesson, we will figure out how the different components in our cluster are communicating with each other and what we need to establish our communication with the worker nodes.

Exploring the AWS ELB

We need a way to access the cluster. So far, we saw that we can, at least, interact with the Kubernetes API. Every time we executed `kubectl`, it communicated with the cluster through the API server. That communication is established through AWS Elastic Load Balancer (ELB). Let's take a quick look at it.

```
aws elb describe-load-balancers
```

The **output**, limited to the relevant parts, is as follows.

```
{
  "LoadBalancerDescriptions": [
    {
      ...
      "ListenerDescriptions": [
        {
          "Listener": {
            "InstancePort": 443,
            "LoadBalancerPort": 443,
            "Protocol": "TCP",
            "InstanceProtocol": "TCP"
          },
          ...
      "Instances": [
        {
          "InstanceId": "i-01f5c2ca47168b248"
        },
          "InstanceId": "i-0305e3b2d3da6e1ce"
        },
        {
          "InstanceId": "i-04291ef2432b462f2"
        }
      ],
      "DNSName": "api-devops23-k8s-local-ivnbim-1190013982.us-east-2.elb.amazonaws.com",
      ...
      "LoadBalancerName": "api-devops23-k8s-local-ivnbim",
      ...
```

Judging from the `Listener` section, we can see that only port `443` is opened, thus allowing only SSL requests.

The three instances belong to managers. We can safely assume that this load balancer is used only for the access to Kubernetes API. In other words, we are still missing access to worker nodes through which we'll be able to communicate with our applications. We'll come back to this issue in a moment.

The entry that matters, from user's perspective, is `DNSName`. That is the address we need to use if we want to communicate with Kubernetes' API Server. Load Balancer is there to ensure that we have a fixed address and that requests will be forwarded to one of the healthy masters.

Finally, the name of the load balancer is `api-devops23-k8s-local-ivnbim`. It is important that you remember that it starts with `api-devops23`. You'll see soon why the name matters.

Looking into the kubectl Configuration

We can confirm that the `DNSName` is indeed the door to the API by examining `kubectl` configuration.

```
kubectl config view
```

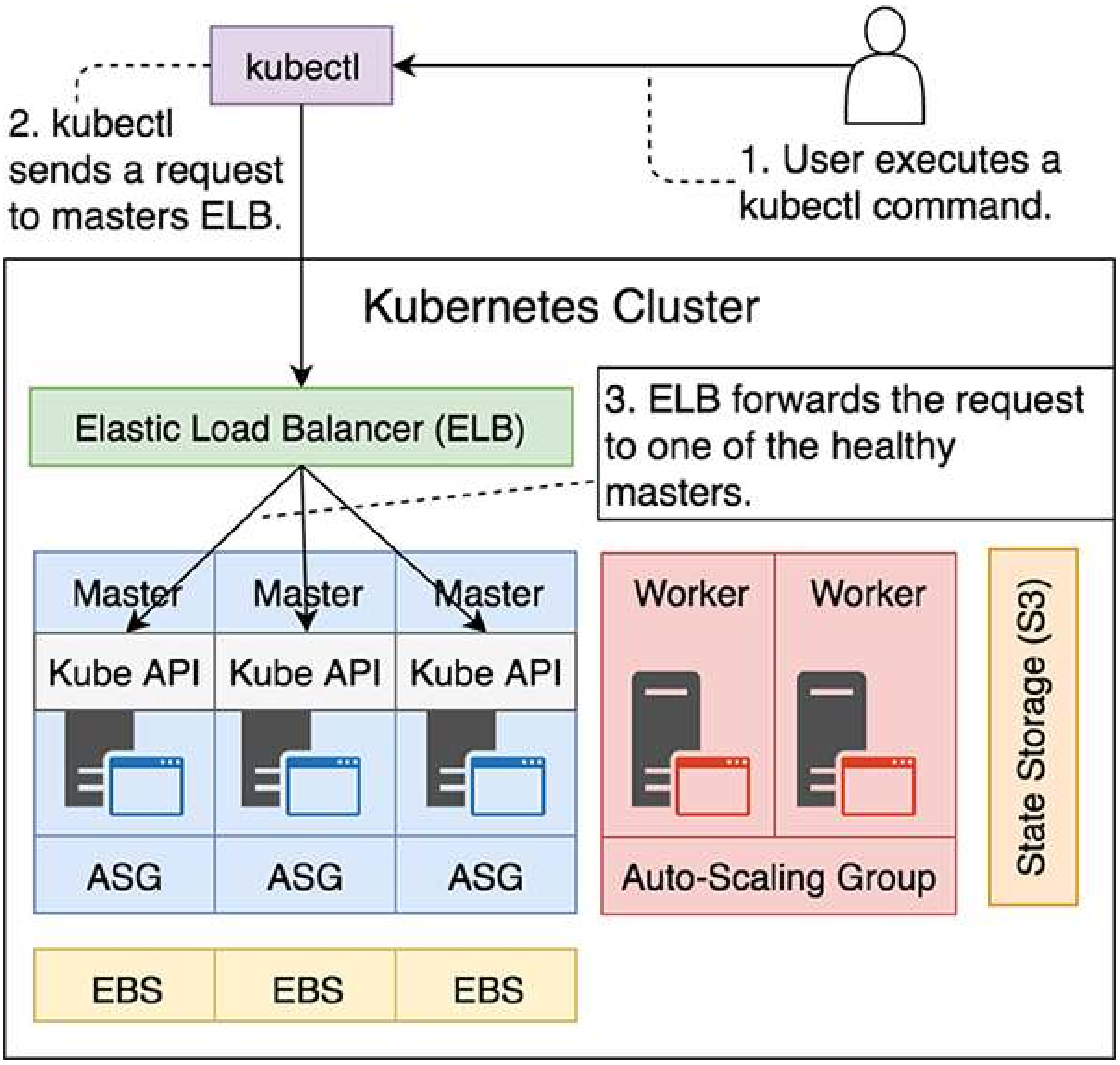
The **output**, limited to the relevant parts, is as follows.

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://api-devops23-k8s-local-ivnbim-1190013982.us-east-2.elb.amazonaws.com
  name: devops23.k8s.local
...
```

```
contexts:
  cluster: devops23.k8s.local
  user: devops23.k8s.local

  name: devops23.k8s.local
...
```

We can see that the `devops23.k8s.local` is set to use `amazonaws.com` subdomain as the server address and that it is the current context. That is the DNS of the ELB.


kubectl
2. kubectl sends a request to masters ELB.
1. User executes a kubectl command.
Kubernetes Cluster
Elastic Load Balancer (ELB)
3. ELB forwards the request to one of the healthy masters.
Master
Master
Master
Kube API
Kube API
Kube API
ASG
ASG
ASG
Worker
Worker
Auto-Scaling Group
State Storage (S3)
EBS
EBS
EBS

Load balancer behind Kuberentes API Server

The fact that we can access the API does not get us much closer to having a way to access applications we are soon to deploy. We already learned that we can use Ingress to channel requests to a set of ports (usually `80` and `443`).

However, even if we deploy Ingress, we still need an entry point to the worker nodes. We need another load balancer sitting above the nodes.

Fortunately, kops has a solution for this.

Fortunately, kops has a solution for this.

In the next lesson, we will add a load balancer to establish communication with the nodes in our cluster.