



Simplified Container orchestration: Service, system and benefits



Containers have become quite popular in recent years and **maintain consistency across platforms**. They have changed the way software is built and maintained.

There are tools developed to extend the functionality to maintain lifecycles of containers. These tools help on

- Placement of right containers on the right node
- Replicate or scaling containers to spread application load across nodes, based on traffic.
- Migrating containers from one host to another in case of any host failures
- Self-healing mechanism. Monitor the status of containers and restart it that goes offline.
- Push rolling updates which ensures the availability of application during deployment.
- Taking care of allocating CPU, Memory, Volumes, IP Address and mapping ports.
- Service Discovery
- Secrets and configuration management

A system which gratifies the above functionality is called **container orchestrator**. Most of the orchestrators work on the declarative method where user can define container state which is required all the time in the cluster while the orchestrator takes care of keeping the containers in the desired state at any instant.

This whitepaper talks about the features of some popular container orchestration tools available in the market such as Kubernetes, Docker Swarm, DC/OS, ECS.

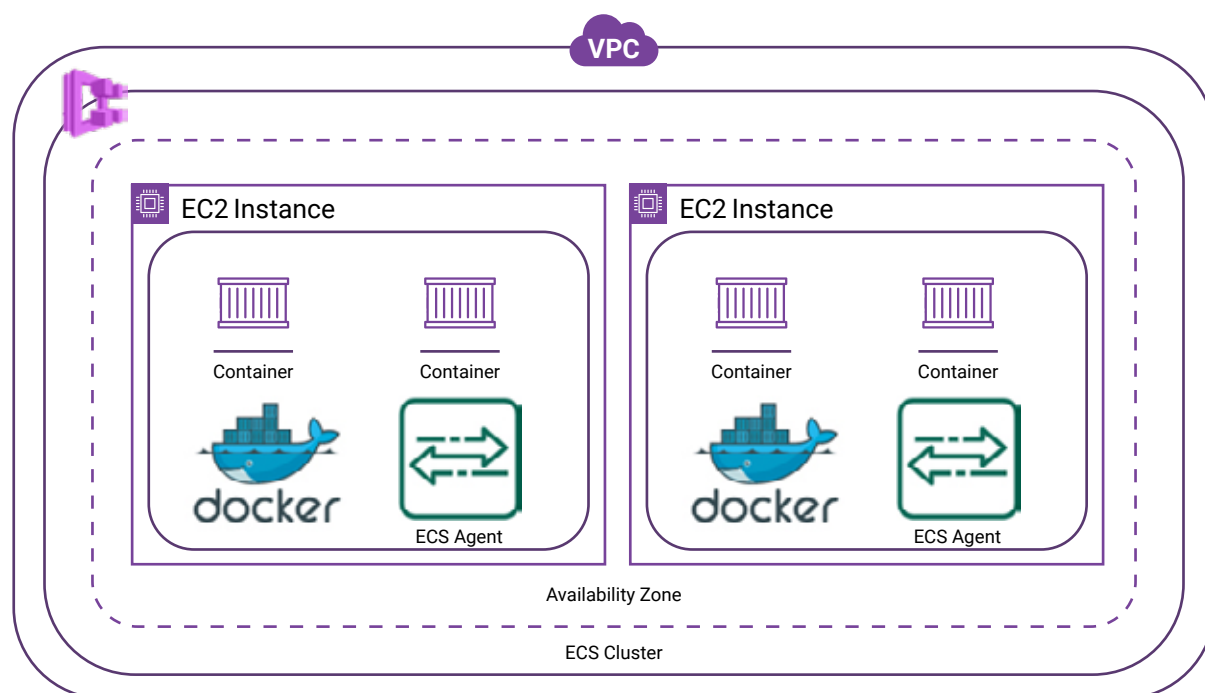


ECS (Elastic Container Service)

What is -----?

Elastic container service is from AWS portfolio. It's an orchestration engine which has been developed by AWS and released to public as open source project called blox (<https://blox.github.io/>). Blox enables you to manage and control containerized applications. You can easily leverage Amazon ECS to completely manage and scale clusters by integrating third party schedulers on ECS.

Architecture





Deployment

You need to provide a task definition (ECS terminology) which describes about your container (image name, resource reservation like cpu, memory, port and log configuration). ECS use this task definition and deploy the containers in the configured VPC.

ECS use the placement strategy to schedule container in the available nodes.

- Binpack
 - Schedule container in node which is having least resource.
- Spread
 - Schedule container evenly based on instance attributes
- Random
 - Schedule instance on any random instances.

We can add custom attributes to instances and make use of it during task creation. So the containers will be created only on instances where the custom attribute is present.





Scalability

- ECS provided with an option called service which helps to define the container specification and allows user to mention how many replicas of containers need to be run in the target cluster.
- Based on the **placement strategy** the ECS will spread the container among the cluster nodes. For eg. If you select the Availability zone based strategy then the containers are scheduled to run across the cluster nodes in all availability zone.
- AWS proprietary auto scaling feature helps a lot in maintaining the required number of containers to fulfill user request. With cloud watch alarm, user can define the metrics based on CPU or Memory that helps to scale {IN/OUT} the tasks based on the load in containers.
- Above mentioned auto scaling will take care of scaling the containers. In order to scale the underlying EC2 instances we need to have a separate scaling policy which monitors the individual EC2 instances in the scaling group.

High Availability

- ECS is a highly available managed service provided by AWS. So administrators don't need to worry about maintaining the ECS.
- Placing EC2 servers spread across the availability zone help to keep the application survive from underlying component failures.



Cost

AWS ECS service comes under free service and we need to pay only for the AWS resources which will be maintained by ECS.

Pricing Models:

i) EC2

EC2 is owned by administrators and the pricing is pretty straight. We only pay for the instances which participate in the ECS cluster.

ii) Fargate

The instances are owned by AWS which makes our life easier. We can concentrate on designing the efficient containers but it comes with a price. Fargate is billed based on the resources we allocate to containers.

Cost calculation = (Running hours of a container * no of vCPU allocated to container * vCPU hourly price) +
(Running hours of a container * memory allocated to container * memory hourly price)

AWS Instance Scheduler

Upon considering the above cost factors you may have a question in mind about what to choose for which scenario? Here is a quick tip:

- a) Do you want your workload to be active all the time and access the underlying instance?
 - Go for EC2 based solution
- b) Is your workload designed to run for short time like batch jobs?
 - Go for Fargate based solution

Security

- ECS is tightly integrated with IAM service that gives us the flexibility to configure which ECS tasks or containers are allowed to access specific AWS services. This can be achieved through IAM roles.

Management

- ECS orchestration service can be handled through well designed AWS console or ECS-CLI which is available in github (<https://github.com/aws/amazon-ecs-cli>). ECS-CLI give us an opportunity to use the Docker compose file which was already developed and used by organizations to define and run multi container application in Docker standalone machine or in Docker swarm cluster.



Capacity

Service quota	Description	Default quota value
Clusters per account	The maximum number of clusters per Region, per account.	10,000
Container instances per cluster	The maximum number of container instances per cluster.	2,000
Services per cluster	The maximum number of services per cluster.	1,000
Tasks using the EC2 launch type per service (the desired count)	The maximum number of tasks using the EC2 launch type per service (the desired count). This limit applies to both standalone tasks and tasks launched as part of a service.	1,000
Tasks using the Fargate launch type or the FARGATE capacity provider, per Region, per account	The maximum numbers of tasks using the Fargate launch type or the FARGATE capacity provider, per Region. This limit applies to both standalone tasks and tasks launched as part of a service.	100
Fargate Spot tasks, per Region, per account	The maximum number of tasks using the FARGATE_SPOT capacity provider, per Region.	250
Public IP addresses for tasks using the Fargate launch type	The maximum number of public IP addresses used by tasks using the Fargate launch type, per Region.	100

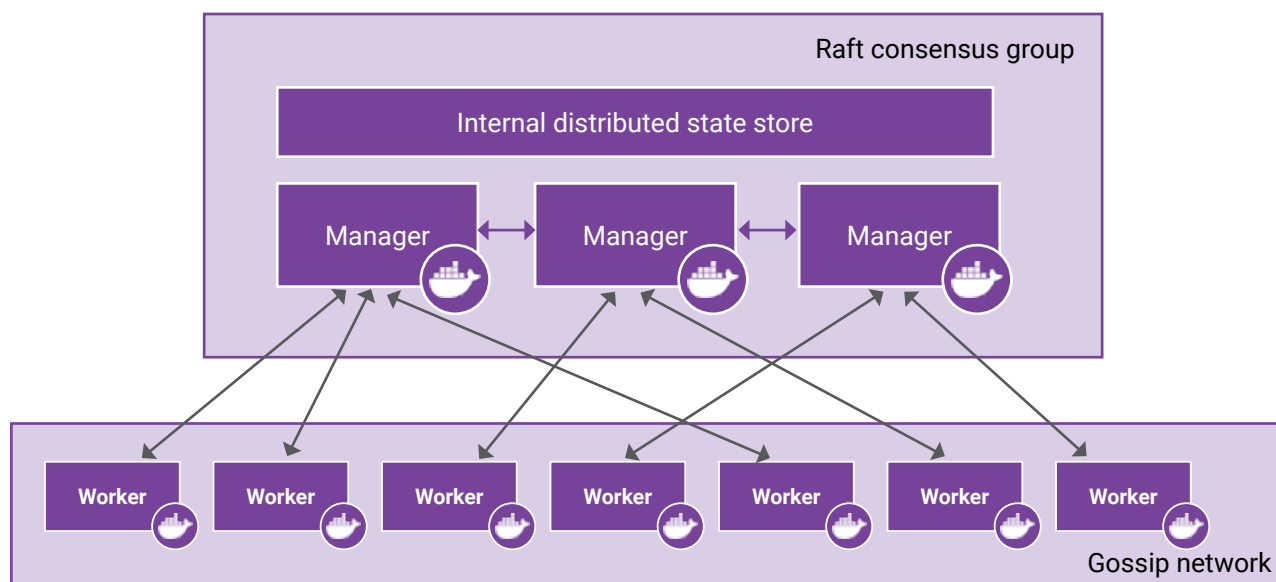


Docker Swarm

What is -----?

Docker swarm is a product of Docker.Inc and it is license free. It comes along with Docker engine as a feature which can be turned ON/OFF. All we need is a Docker cli utility to interact with Docker swarm cluster. It is a light weighted and one of the simple orchestrators in the market comparing to others.

Architecture





Deployment

- Docker swarm is equipped with scheduler as its internal component. No need to deploy any additional services to enable the scheduling functionality.
- In Docker swarm cluster, we can create an entire application stack through a compose file. Docker swarm accepts compose file of version 3.0 or higher. Sample compose file:

```
version: '3'

services:

  web:

    image: nginx:latest

    ports:

      - "8000:8000"

  redis:

    image: redis:alpine
```

- As soon as the context gets uploaded, the scheduler checks for available nodes and validate whether the available resource is enough to run the containers and schedule the tasks. If no resource is available then Docker swarm keeps the request in pending state until enough resource is freed up or new node get attached to the cluster.
- Custom labels' feature help to place the containers in specific nodes. Creating custom label like `env="development"` in nodes which belongs to the development stage helps us to restrict the deployment of containers only to development environment.



Scalability

- Dockers have inbuilt mechanism which enables users to scale-IN/OUT the number of replicas of deployed stack or services. But, this needs to be done manually.
- Docker doesn't have any monitoring functionality to analyze the load among the containers and makes scaling decision based on load.
- We can integrate Docker swarm with other third party monitoring system to analyze the system load and trigger scaling by accessing Docker CLI or Docker API.
- Running Docker swarm in cloud will give the flexibility to manage the worker nodes based on the cloud native auto scaling feature.

High Availability

- Adding multiple master and worker nodes to Docker swarm will make the infrastructure highly available. Docker recommends maintaining the master nodes which adhere to quorum count. Literally it should be >50%. For eg, if we have two master nodes and if any one goes down then we have exactly 50% of the highly available resource which will fail the quorum. In this scenario having three master nodes is the perfect design which maintains the quorum in the environment.
- Failed to maintain the quorum will not affect the running containers. But users won't be able to make any changes to the cluster.
- Docker in built feature "HEALTH CHECK" helps to monitor the application which is running inside the container. If the check fails for a defined number of times then the container is marked as UNHEALTHY and it will get re-started.

Cost

- Docker community edition is free. So the cost is completely relying on the underlying infrastructure like Virtualization (VMware) or cloud based VM instances.
- If user wants a web console to maintain the cluster infrastructure and private registry, then enterprise edition license is the one-stop shop they need to approach. The enterprise edition will give a wonderful UI console where administrators can do user management (Role Based Access Control), deploy and maintain stacks and cluster management like add/remove master and worker nodes.



Management

- Docker CLI is quite enough to maintain the entire container but still Docker gives an option to use its UI by purchasing a Docker Enterprise Edition.
- Not only the UI , with the enterprise edition license, user will get private Docker trusted registry called DTR which can be installed in user infrastructure and keep the image much more safe by isolating it from public access.

Security

- Additional nodes to Docker swarm infrastructure can be added only by specifying the token belonging to the role (Master/Worker). Tokens can be rotated in case of details got compromised.
- Docker creates a CA certificate authority when user initiates the Docker swarm functionality. Each node will get its own certificate to establish secure communication with master.
- The nodes in a swarm use mutual Transport Layer Security (TLS) to authenticate, authorize, and encrypt the communications with other nodes in the swarm.
- No inbuilt RBAC mechanism in the community edition. The same can be achieved through the enterprise edition.
- DTR inbuilt capability to scan the images helps to identify the vulnerability in the uploaded images. DRT automatically download the vulnerability database and keep the vulnerability footprints up-to-date.

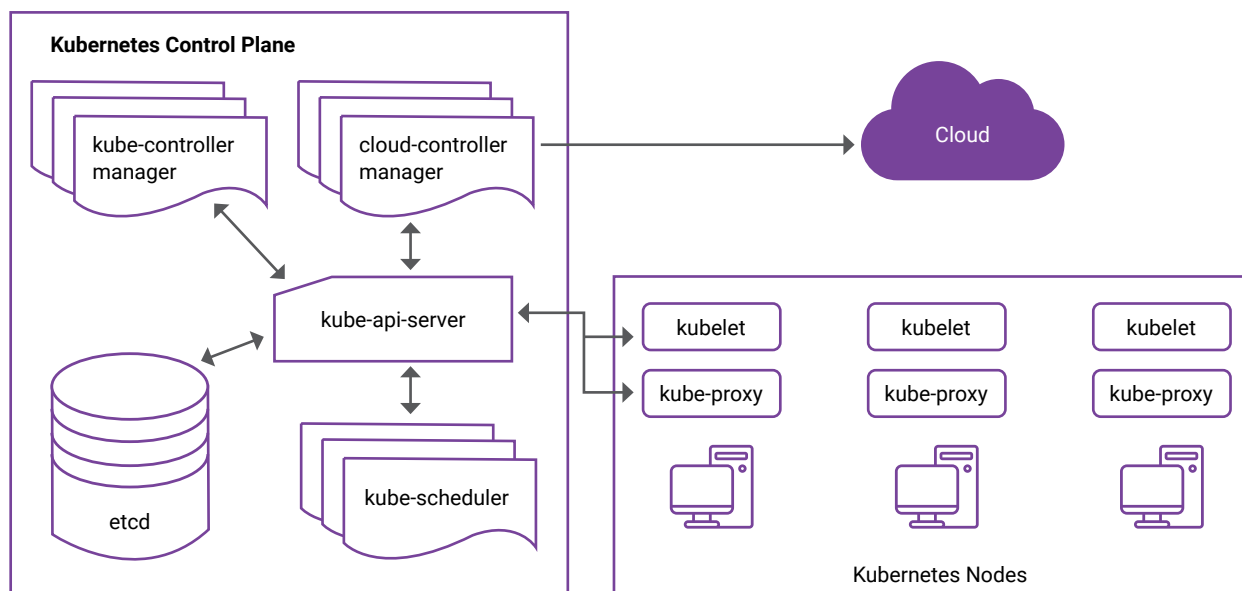


Kubernetes

What is -----?

- Kubernetes is one of the popular orchestration system which was initially developed by Google and donated to Cloud Native Computing Foundation (CNCF) as open source. Popular cloud service providers Amazon, Azure, Google adapted kubernetes and published as a service in their portfolio (Elastic Kubernetes Service, Azure Kubernetes Service, Google Kubernetes Service). Users can either build the kubernetes from scratch in their on-prem datacenter or opt for managed service from any one of the cloud service provider.
- Utilizing managed kubernetes service from cloud vendor have both pros and cons
 - Pros
 - Kubernetes is tightly integrated with other cloud services like IAM, Storage, Monitoring which makes life easier in terms of integration.
 - Cloud vendor takes care of spanning multiple masters to ensure that k8s service is always available which reduced our maintenance efforts.
 - cons
 - Kubernetes master is completely managed by the vendor and we will not get access to the master environment.
 - We can't shut down the master, which is not a cost effective option. Environments built for on-demand request (dev) either should destroy the k8s environment or leave it in running state bearing the cost. At the time of writing this article only AWS EKS charge for master nodes.

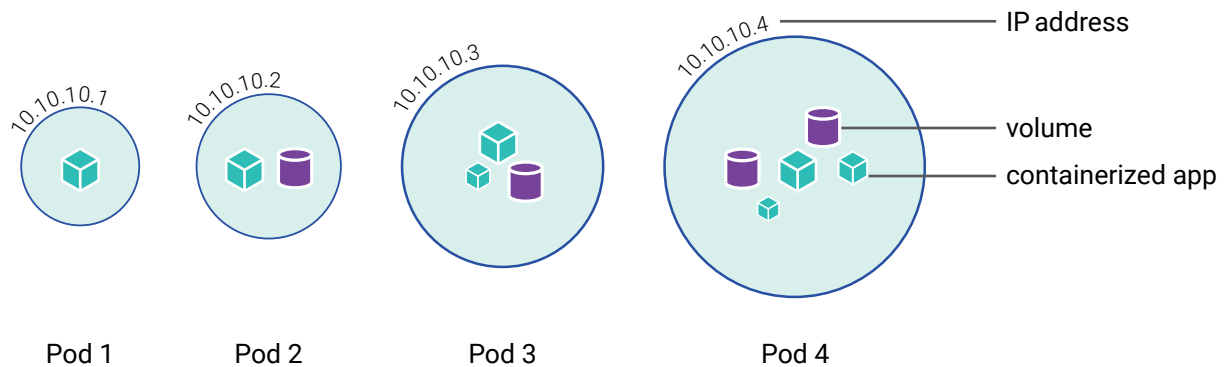
Architecture





Deployment

- Kubernetes uses a different approach to bundle multiple containers in the form of PODS. The container which we deploy in k8s goes inside a unique pod. Multiple containers inside the pod share the same network and storage. This nature of PODS makes the data transmission easy between the containers inside the pod.



- K8s accepts the object definition file in the format of YAML. This helps to keep the changes to k8s cluster tracked via version control.





POD Definition	Deployment Definition	Service Definition
apiVersion: v1 kind: Pod metadata: name: memory-demo namespace: mem-example spec: containers: - name: memory-demo-ctr image: polinux/stress resources: limits: memory: "200Mi" requests: memory: "100Mi" command: ["stress"] args: ["-vm", "1", "-vm-bytes", "150M", "-vm-hang", "1"]	apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deployment spec: selector: matchLabels: app: nginx replicas: 2 template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx:1.14.2 ports: - containerPort: 80	apiVersion: v1 kind: Service metadata: name: hello spec: selector: app: hello tier: backend ports: - protocol: TCP port: 80 targetPort: http

- Kubectl cli utility helps to interact with a k8s cluster. User can pass the yaml definition file to kubectl to create/modify/delete the objects in the kubernetes.
- As an alternative, users can use package manager utility like HELM to simplify the deployment process. Like Linux package managers (YUM, APT) HELM helps to package the application definitions with versioning for kubernetes and makes life easier when it comes to deployment.



Scalability

- K8s is a master-slave architecture based system where we can add more worker nodes based on the payload. K8s on cloud makes this very simple by adopting best in breed auto scaling mechanism. Based on the metrics (cpu, memory) the nodes will get scaled IN/OUT which not only making the k8s environment more elastic, it controls the cost by eliminating the unwanted nodes during ideal or low traffic situations.
- On another hand scaling of pods can be achieved either manually through kubectl cli or inheriting horizontal pod autoscaler (hpa) inbuilt feature. By default HPA monitors the cpu utilization of pods in a deployment and make decision about scaling based on the threshold configured in the HPA object.

High Availability

- K8s maintain the cluster state in Etcd key value store which is the heart of entire orchestration system. K8s frequently query all the nodes in the cluster and update the state in Etcd. Keeping Etcd highly available is going to be a priority one.
- Like Docker swarm, k8s master node can be scaled to maintain cluster in a highly available state. Managed k8s service from cloud vendors eliminates the responsibility of managing master nodes from our hand. Cloud vendor spin up multiple master nodes in different availability zone to ensure that k8s administration is always possible.
- Kubernetes master component "Replication controller" watch each deployment and replica set to ensure that desired level pods are running in the cluster. It spans a new pod if the total count of pod is not matched with values mentioned in the replica set.
- K8s liveness and readiness probe monitors the containers which is running inside a pod and take action based on the monitoring output. Readiness probe avoids sending user request to pod which has been recently started/restarted where the actual process inside the container is not yet ready to respond back to user request. Liveness probe monitors the process running inside the containers continuously to take action if process is not in healthy state for some reason like application crash or process hung.
- These self- healing mechanisms ensure that user experience is not impacted.



Cost

- K8s is available under open source license. So it won't burn the pocket. But underlying infrastructure should be designed in such a way to avoid over provisioning of worker nodes which leads to negative ROI over a period of time.
- Apart from AWS EKS other two popular cloud vendors AZURE and GOOGLE provides master node control plane as free of cost. We only pay for the instances attached with k8s cluster.

Security

- Etcd component in the master server is only accessible through API server. Which ensures that it's can't be tampered by other cluster components.
- Cluster component like Kubelet, kube-proxy, control pane, replication controller are all interact with API server through PKI infrastructure which ensures only the components with valid certificate is allowed to establish the communication.
- K8s comes with a rigid RBAC mechanism where authorizations to users/teams are handled through {role|clusterrole} bindings. Roles specifies what action a user can do on a specific namespace or all namespace(clusterrole) and role-binding binds a role with a user which enables the user required access.



Management

- K8s dashboard project giving us a pretty decent WEBUI which is sufficient to observe what's going on in the cluster. Cluster operations like creating deployments, rolling updates, scaling pods are possible through this WEBUI.
- The Dashboard UI is not deployed by default and configuring it not a straight forward action. The dashboard is not publicly accessible by default; we need to establish a tunnel from machine where the kubectl utility is available with the kubernetes cluster to do port forwarding.

Kubectl cli is a command line tool for controlling Kubernetes clusters. We can control the kubernetes objects either through one liner command or we can craft that as a valid yaml file and can pass it to kubectl utility as an argument.

Capacity

MAX Nodes	: 5000
MAX pods	: 150000
MAX containers	: 300000
MAX pod per node	: 100





DC/OS

DataCentre Operating System

What is -----?

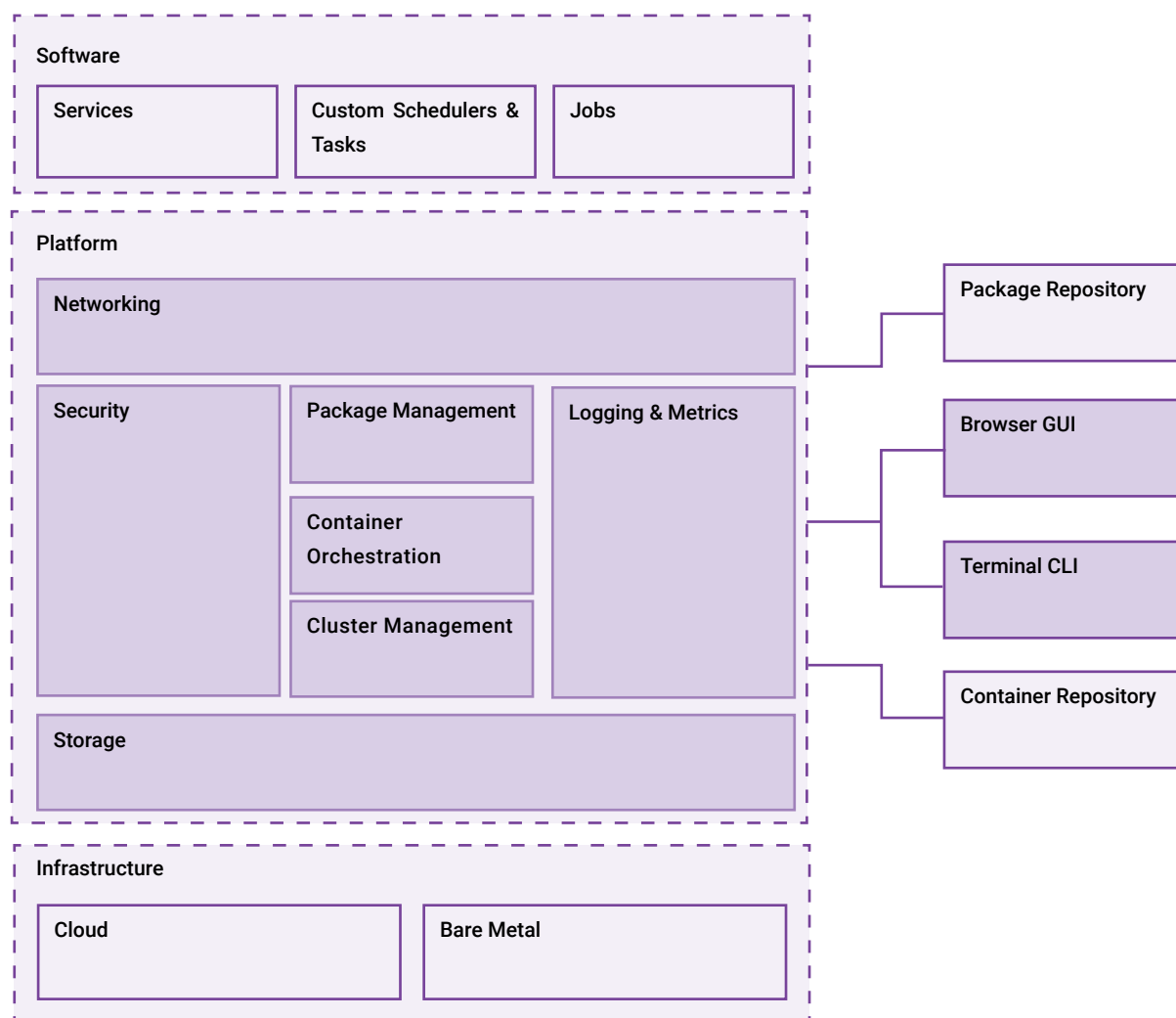
- It's primarily called as datacenter operating system which actually abstracts the entire datacenter resources into a giant resource pool. It is really a platform which manages compute clusters and being sometimes described as distributed kernel. It's been designed with an objective in mind about managing both containerized and non-containerized workloads in a distributed way.
- DCOS is a master-slave architecture implementation where master takes care of abstracting the underneath resources like bare metal servers or VM's or cloud instances and brings everything under a resource pool. The workloads are scheduled in the slave nodes.
- DCOS comes with most of the functionality which a orchestration system need,
 - Its abstracts the underneath layer
 - Tracks all the workloads and resource consumption
 - Isolating the process from others
 - Restart workloads in the situation of node or task failure
- It's not initially designed to maintain containers but later an upstream project called MARATHON framework enabled the functionality. DCOS + MARATHON = Better orchestration system.
- If the environment demands a large distributed system like Hadoop, Apache Spark, along with containerized application then DCOS will be best fit to maintain the entire infrastructure.





Architecture

Mesosphere DC/OS





Deployment

- We can use web UI to deploy a single or multi container like POD in DCOS environment or we can define our deployment in declarative way using json format and can deploy the applications.

Scalability

- DCOS have the capability to handle thousands of nodes but it lack with auto scaling feature. In another angle this is out of scope of DC/OS. What if the DC/OS manages the bare metal server?
 - There are open source projects available which can be added to DCOS environment to scale the underlying servers to scale-IN/OUT based on the metrics it collects. But it needs the DC/OS agents should be running in a cloud environment.
 - In another way we can make use of the out-of-box autoscaling functionality in cloud to scale the nodes IN/OUT based on the cloud monitoring metrics.
- The same goes to containers. Marathon framework allow us to scale the tasks manually but unlike Kubernetes (HPA) we don't have an option yet to make the containers scale based on the resource metrics. We need to use some kind of monitoring system to validate the resource consumption and have to use the marathon API to scale IN/OUT the containers based on the need.



High Availability

- DCOS has been designed to handle large distributed system. So it's hard to doubt about its high availability. DC/OS and marathon high availability are supported through Zookeeper. It helps to elect DC/OS and Marathon leaders to maintain cluster state.
- It's intelligent enough to monitor the underlying infrastructure and schedule the task based on the infrastructure health. If an underlying node fails then the tasks are re-scheduled to other nodes.
- DC/OS provides health check options to periodically check the distributed containers across the cluster. If a container reaches the maximum number of consecutive failures then the system will kill the task and restart it.

Cost

- DC/OS comes with two variant, Open source and Enterprise edition.

Security

- DCOS provides variety of authentication mechanism like LDAP, single sign-on with SAML and even it gives channel to integrate with multiple identity vendors which simplify the user management and avoid duplicate work.
- Out of box feature "SECRETS" help to decouple the hardcoded application relevant sensitive configuration and kept it in encrypted during transmission and in rest.
- The nodes participating in the cluster are communicated through SSL which ensures that the communication is secured.

Management

- It comes with a good featured web UI and CLI which helps to monitor and administrate the entire DCOS infrastructure.
- We can deploy a single and multi-container through UI and scale it manually based on the demand.



Conclusion

- If your application architecture is not complex and consists of only few micro service then you can leverage the capability of AWS ECS
- If you want to manage the underlying infrastructure and need a simple orchestration system then Docker Swarm is the best approach, but this comes with no auto scaling.
- If your application architecture consists of more number of micro services and want to scale up to thousand node or if you want to leverage the large community support or if you want to run your application in cloud and don't want to take the overhead of managing the master nodes, then you can prefer cloud based Kubernetes services like AKS, EKS and GKE.
- If you want an orchestration system which takes care of non-container and container workloads, if you want a single pane of system to handle your entire infrastructure efficiently along with container orchestration then you can try DC/OS.

About Us



Aspire Systems is a global technology services firm serving as a trusted technology partner for our customers. We work with some of the world's most innovative enterprises and independent software vendors, helping them leverage technology and outsourcing in our specific areas of expertise. Our core philosophy of "Attention. Always." communicates our belief in lavishing care and attention on our customer and employees.

[For more info contact](#)

info@aspiresys.com or visit www.aspiresys.com

NORTH AMERICA
+1 630 368 0970

POLAND
+44 203 170 6115

INDIA
+91 44 6740 4000

MIDDLE EAST
+971 50 658 8831

EUROPE
+44 203 170 6115

SINGAPORE
+65 3163 3050