Microservices deployment and scaling with Kubernetes

VietKubers (Vietnam Kubernetes Community Group – https://vietkubers.github.io)

Dao Cong Tien (tiendc@gmail.com)

AGENDA

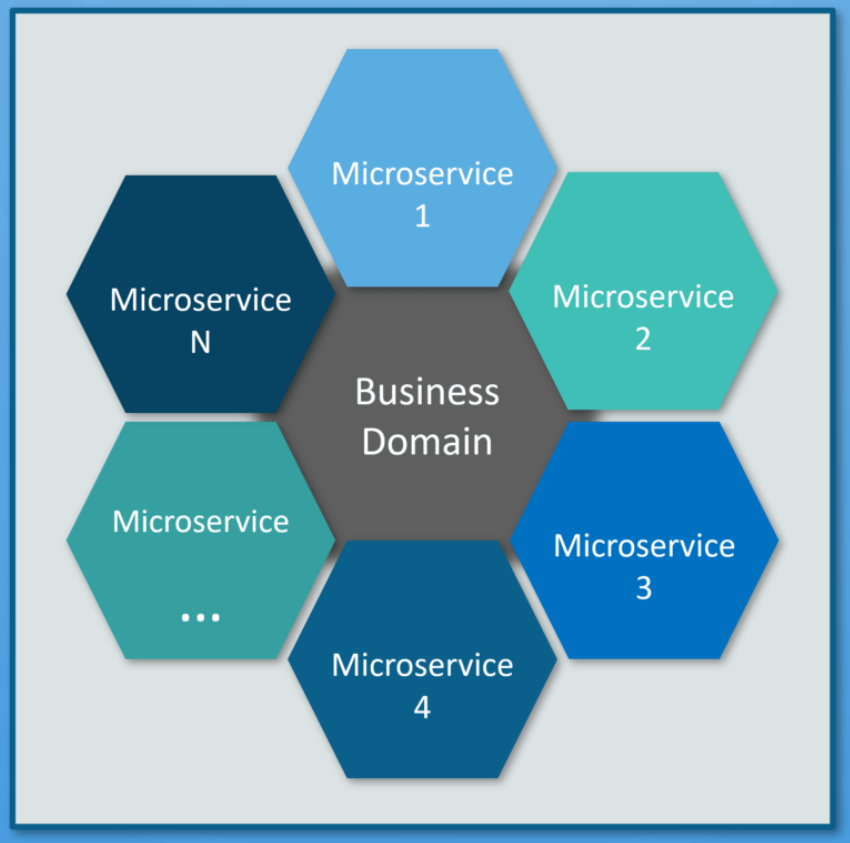
1. Microservices
2. Microservices Deployment with Kubernetes
3. Microservices Scaling with Kubernetes
4. DEMO
5. Q&A

AGENDA

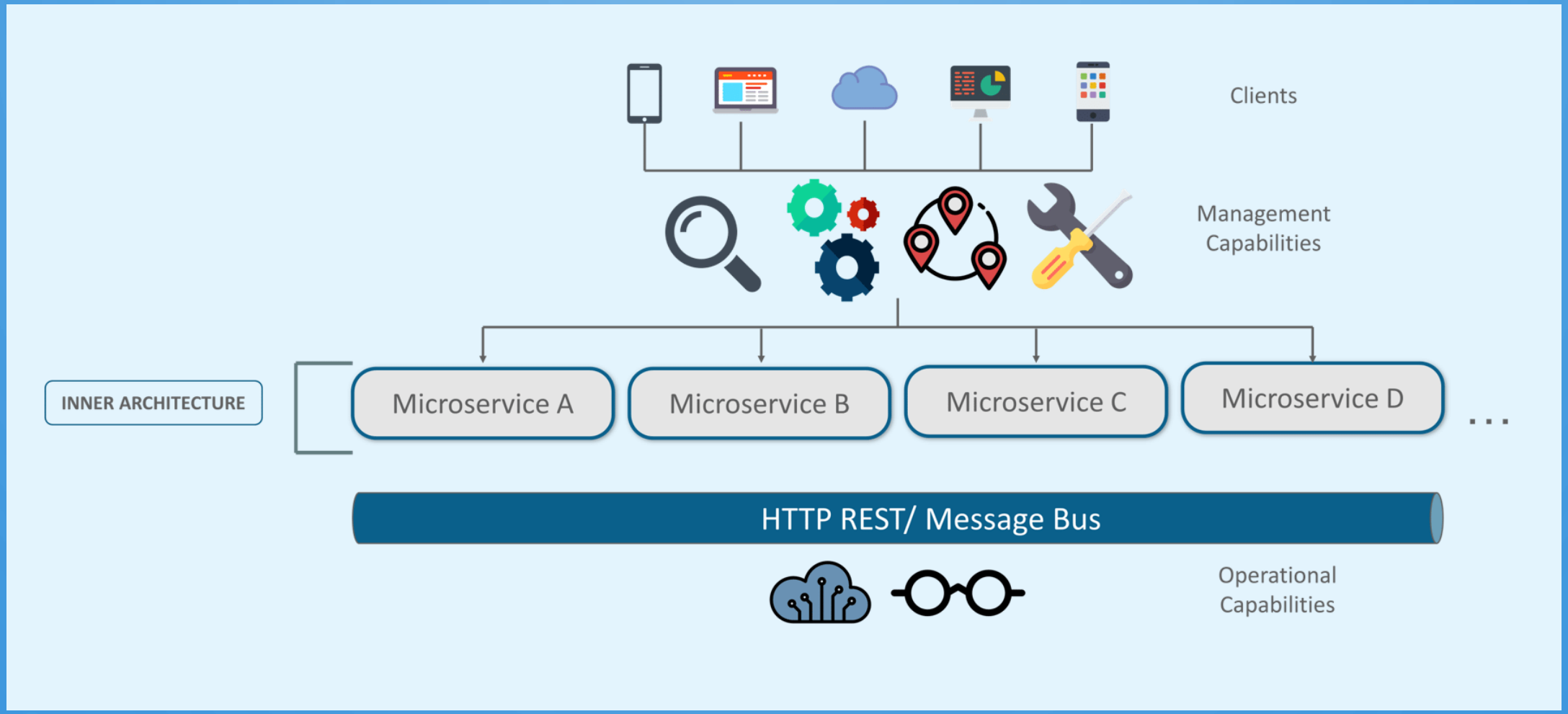
1. Microservices
   - What is Microservices
   - Microservices vs. Monolithic architecture
2. Microservices Deployment with Kubernetes
3. Microservices Scaling with Kubernetes
4. DEMO
5. Q&A

WHAT IS MICROSERVICES

- is an architectural style that structures an application as a collection of small autonomous services, modeled around a business domain.



WHAT IS MICROSERVICES



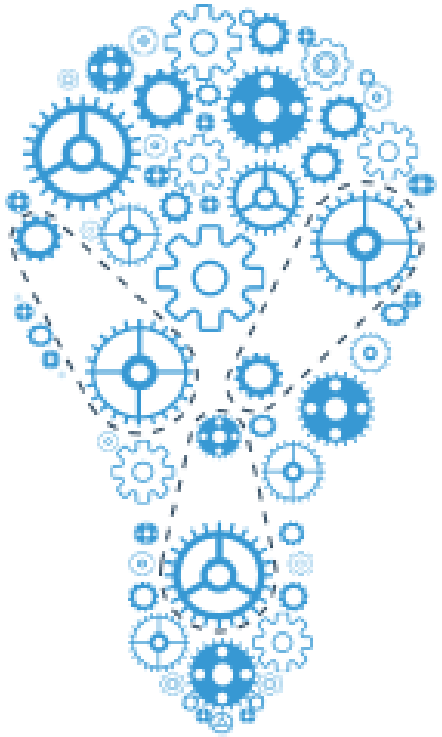
MICROSERVICES VS. MONOLITHIC ARCH

1990s and earlier

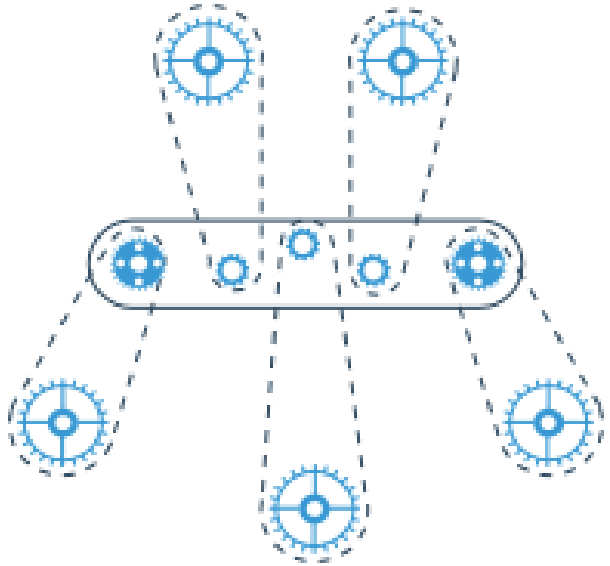
2000s

2010s

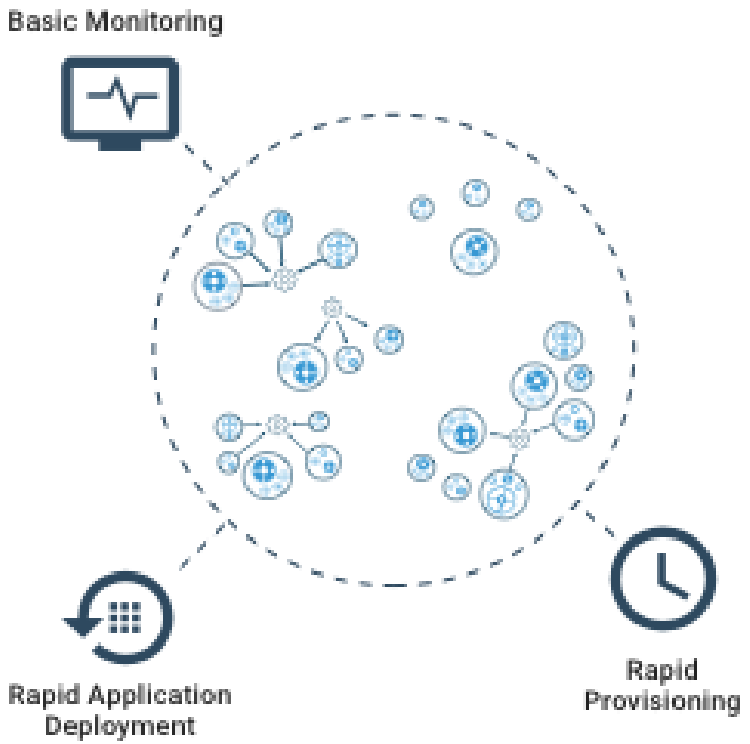
Monolithic Architecture



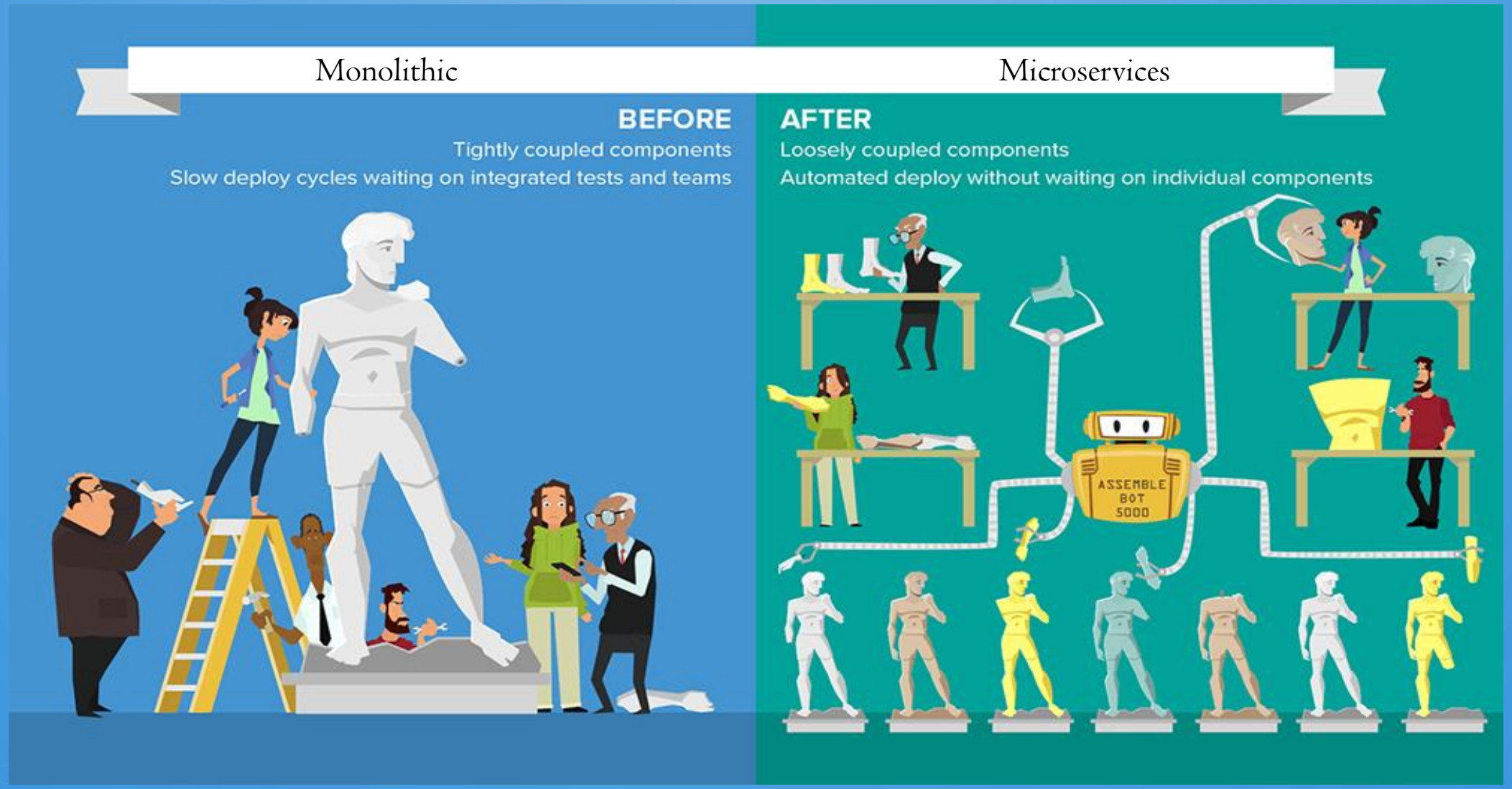
Traditional SOA



Microservices



MICROSERVICES VS. MONOLITHIC ARCH

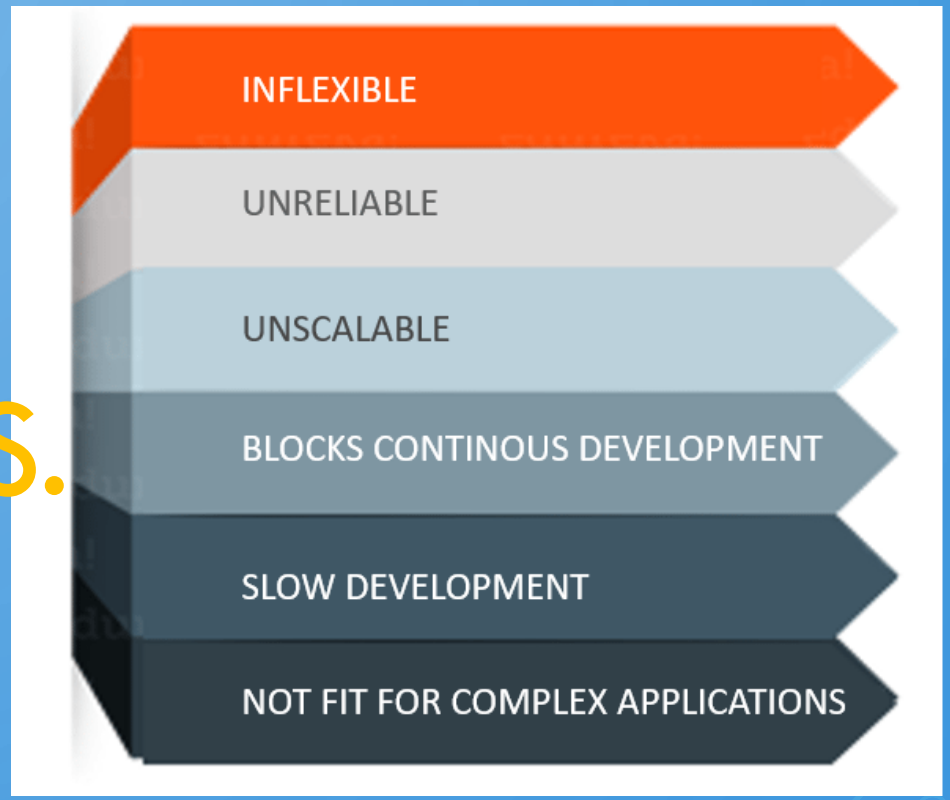


MICROSERVICES VS. MONOLITHIC ARCH

Microservices

VS.

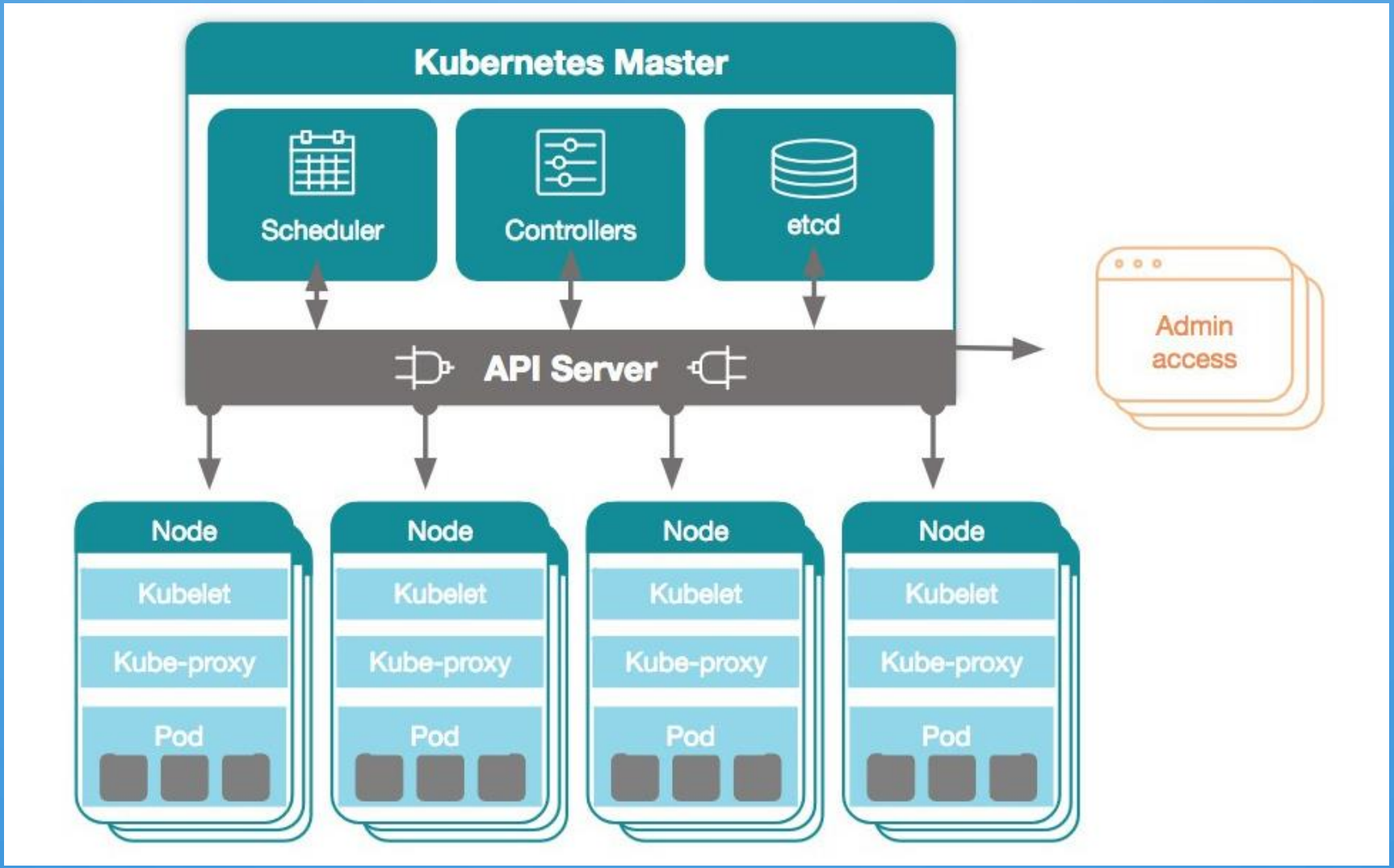
Monolithic



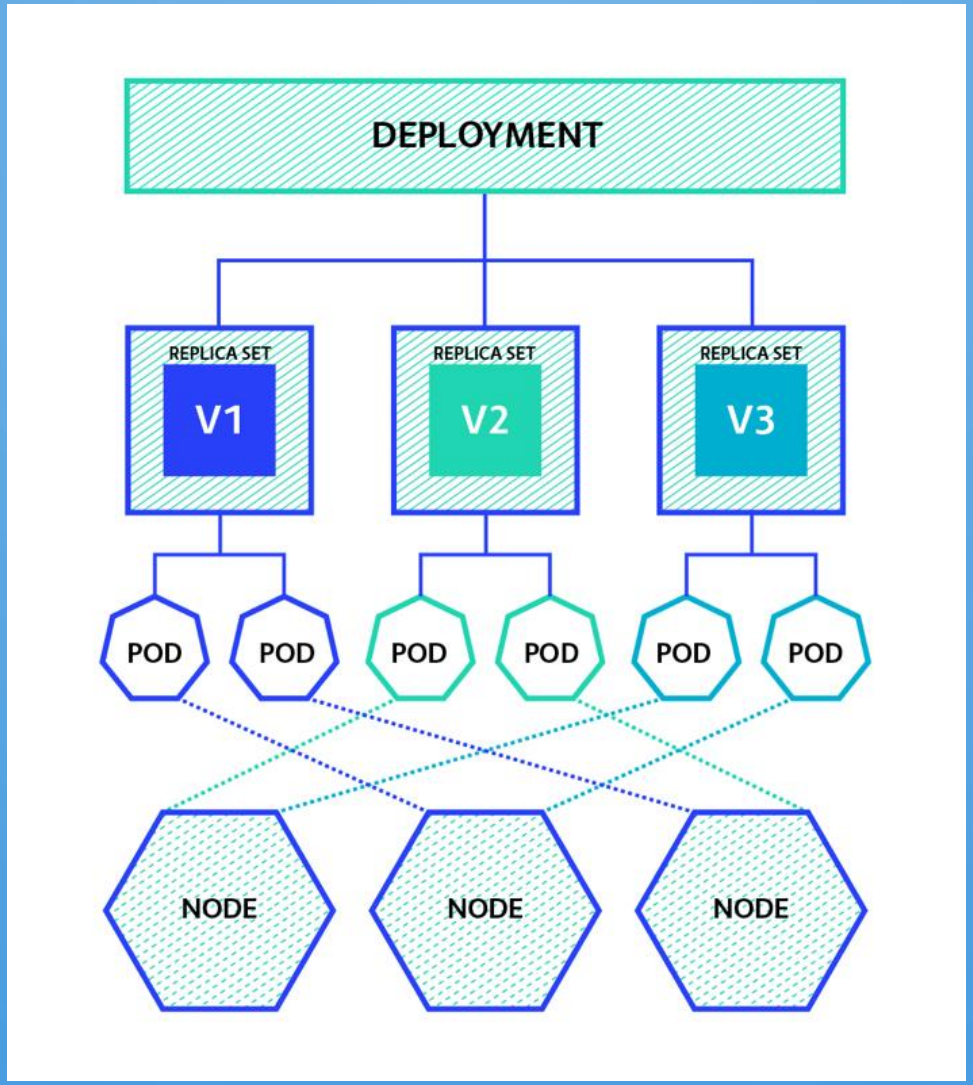
AGENDA

1. Microservices
2. Microservices Deployment with Kubernetes
   - Kubernetes architecture
   - Kubernetes deployment
   - Kubernetes service
   - Kubernetes replicas
3. Microservices Scaling with Kubernetes
4. DEMO
5. Q&A

KUBERNETES ARCHITECTURE

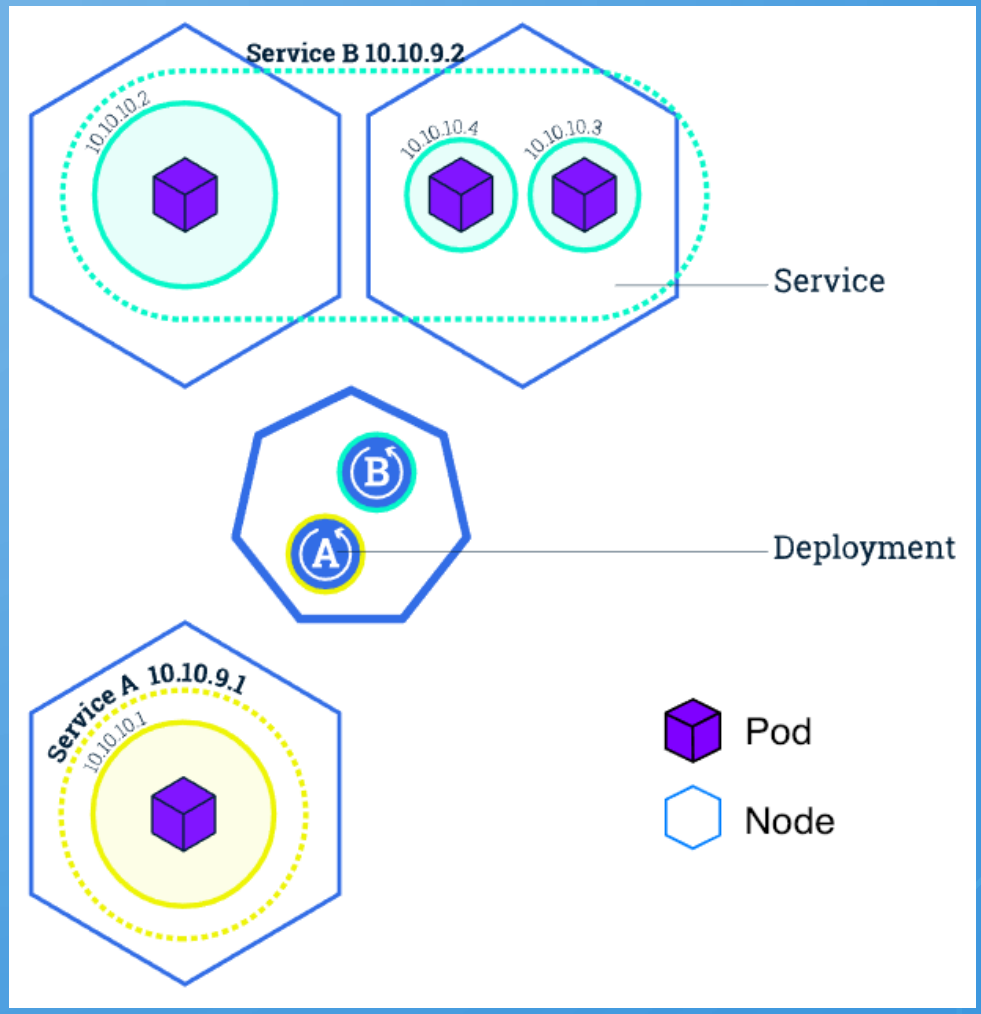


KUBERNETES DEPLOYMENT



KUBERNETES SERVICE

- A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them
  - Without Kubernetes Service, a Pod is unable to talk to other Pods or a user is unable to interact with the service from outside of the cluster

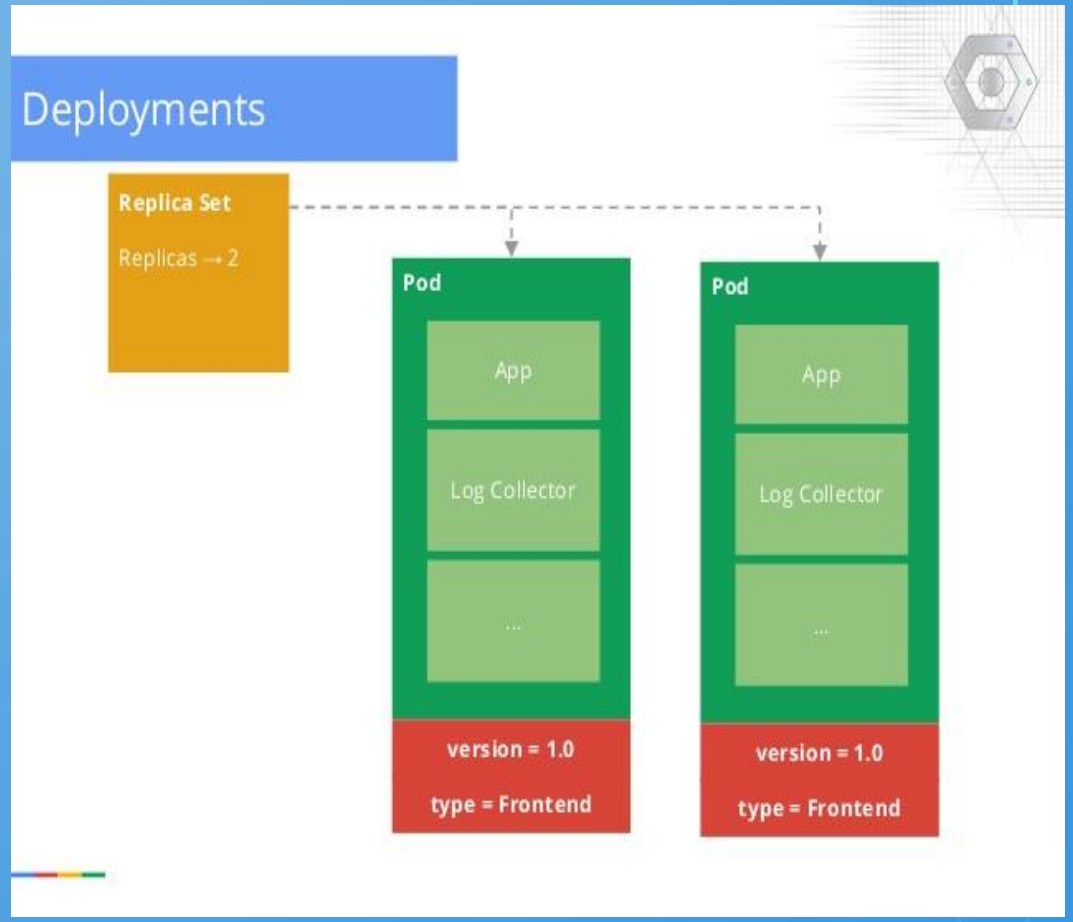


KUBERNETES SERVICE TYPES

- ClusterIP
  - The service only reachable from within the cluster
- NodePort
  - Exposes the service on each Node's IP at a static port (the NodePort)
  - Accesses the service by:
    - Using ClusterIP from within the cluster
    - Accessing from outside the cluster, by requesting <NodePort>
- LoadBalancer
  - Exposes the service externally using a cloud provider's load balancer.
  - Accesses the service by:
    - Using ClusterIP, or NodePort
    - Accessing from the external network
- ExternalName
  - Maps the service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value

KUBERNETES REPLICAS

- The basic idea: you have multiple, identical copies of your pod running, potentially on different nodes in the cluster
  - Kubernetes can distribute the load between Pods
  - The controllers can (re-)launch pods as they see fit, reacting to the health of the containers running in the pod
  - If the load increases, you may launch even more lovely clones of your pod
  - If a node in the cluster fails, the controllers will find a new home for your clones



AGENDA

1. Microservices
2. Microservices Deployment with Kubernetes
3. Microservices Scaling with Kubernetes
   - Cluster scaling
   - Horizontal Pod scaling
   - Vertical Pod scaling
4. DEMO
5. Q&A

KUBERNETES SCALING

- Story: what happens when you build a service that is even more popular than you planned for, and run out of compute?
- Solution:
  - Manual scaling?
  - Auto scaling?

KUBERNETES AUTO-SCALING

- Cluster auto-scaling
  - automatically adjusts the size of the Kubernetes cluster when one of the following conditions is true:
    - there are pods that failed to run in the cluster due to insufficient resources,
    - there are nodes in the cluster that have been underutilized for an extended period of time and their pods can be placed on other existing nodes
  - Supports Google Compute Engine (GCE), Google Container Engine (GKE), AWS, Azure, Alibaba…

KUBERNETES AUTO-SCALING

- Horizontal Pod auto-scaling (HPA)
  - Horizontal Pod Autoscaler automatically scales the number of pods in a replication controller, deployment or replica set based on observed CPU utilization (or, with beta support, on some other, application-provided metrics).
  - Example:
    - $ kubectl autoscale deployment <deploy> --cpu-percent=70 --min=1 --max=10

KUBERNETES AUTO-SCALING

- Vertical Pod auto-scaling (VPA)
  - sets the requests automatically based on usage, allows proper scheduling onto nodes
  - reacts to most out-of-memory events
  - Pods will be restarted when scaling
  - does not change resource limits

AGENDA

1. Microservices
2. Microservices Deployment with Kubernetes
3. Microservices Scaling with Kubernetes
4. DEMO
5. Q&A

DEMO – MICROSERVICES DEPLOYMENT

- A simple TODO application (with 2 services)
- https://github.com/vietkubers/demo/tree/master/microservices-demo-todo-app

DEMO – MICROSERVICES DEPLOYMENT

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dep-users
spec:
  selector:
    matchLabels:
      run: dep-users
  replicas: 1
  template:
    metadata:
      labels:
        run: dep-users
    spec:
      containers:
      - name: todoapp-users
        image: docker.io/tiendc/todo-app-users:1.0
        ports:
          - containerPort: 5000
        envFrom:
        - configMapRef:
            name: my-config
        resources:
          limits:
            cpu: "1"
            memory: 100Mi
          requests:
            cpu: 500m
            memory: 50Mi
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dep-todo
spec:
  selector:
    matchLabels:
      run: dep-todo
  replicas: 2
  template:
    metadata:
      labels:
        run: dep-todo
    spec:
      containers:
      - name: todoapp-todo
        image: docker.io/tiendc/todo-app-todo:1.0
        ports:
          - containerPort: 5001
        envFrom:
        - configMapRef:
            name: my-config
        resources:
          limits:
            cpu: "1"
            memory: 100Mi
          requests:
            cpu: 500m
            memory: 50Mi
```

DEMO – MICROSERVICES DEPLOYMENT

- After launching app
  - Lists all users: http://<NodeIp>:32000/users
  - Lists all todo: http://<NodeIp>:32001/todo
  - View a user: http://<NodeIp>:32000/users/<username>
  - List all Todo of a user: http://<NodeIp>:32000/users/<username>/todo

```yaml
kind: Service
apiVersion: v1
metadata:
  name: svc-users
  labels:
    run: dep-users
spec:
  type: NodePort
  selector:
    run: dep-users
  ports:
  - protocol: TCP
    port: 5000
    targetPort: 5000
    nodePort: 32000
```

```yaml
kind: Service
apiVersion: v1
metadata:
  name: svc-todo
  labels:
    run: dep-todo
spec:
  type: NodePort
  selector:
    run: dep-todo
  ports:
  - protocol: TCP
    port: 5001
    targetPort: 5001
    nodePort: 32001
```

DEMO – MANUAL SCALING

- Scale up (2 Pods to 3 Pods)
- `# kubectl scale --replicas=3 deployment/dep-users`
- Scale down (2 Pods to 1 Pod)
- `# kubectl scale --replicas=1 deployment/dep-todo`

DEMO – AUTO SCALING

- Must install metrics-server for using HorizontalPodAutoscaler
- https://github.com/kubernetes-incubator/metrics-server

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-users
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: dep-users
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 80
```

AGENDA

1. Microservices
2. Microservices Deployment with Kubernetes
3. Microservices Scaling with Kubernetes
4. DEMO
5. Q&A

REFERENCES

- https://www.edureka.co/blog/what-is-microservices/
- https://kubernetes.io/docs/
- And some other images on the internet…

THANK YOU