< 쿠버네티스 리소스 >

쿠버네티스에서는 「리소스」를 등록하는 것으로 컨테이너의 실행과 로드밸런서의 작성을 비동기로 실행됩니다. 쿠버네티스가 취급하는 리소스의 종류는 여러가지가 있으며, 하기의 표에서 API Reference 가 공개되어 있습니다.

또한 리소스는 크게 나누어 다음의 5 종류가 있습니다.

종류	개요
Workloads 리소스	컨테이너의 실행에 관한 리소스
Discovery & Load Balance 리소스	컨테이너를 외부에 노출하기 위한 엔드 포인트를 제공하는 리소스
Config & Storage 리소스	설정 / 기밀정보 / Persistent Volume 등에 관한 리소스
Cluster 리소스	보안 및 쿼타(Quota) 등에 관한 리소스
Metadata 리소스	클러스터 내의 다른 리소스를 조작하기 위한 리소스

이 가운데 애플리케이션 개발자가 특히 이용하는 것은 Workloads / Discovery & Load Balance / Config & Storage 인 3 종류입니다.

1. Workloads 리소스

첫번째인 Workloads 리소스는 클러스터 상에 컨테이너를 가동시키기 위해 이용하는 리소스입니다. 내부적으로 사용되고 있는 것을 제외한 이용자가 직접 조작하는 것은 전부 8 종류의 Workloads 리소스가 있습니다.

- **Pod**

한 개 이상의 컨테이너로 구성된 컨테이너 집합입니다.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

- **ReplicationController**

Pod가 스펙과 동일하지 않게 존재할 경우 스펙과 동일한 상태가 되도록 관리합니다. 그러나 쿠버네티스 버전이 올라감에 따라 레플리케이션 컨트롤러는 더 이상 사용 되지 않으며 (deprecated) , 그 대신 레플리카셋이사용되고 있습니다.

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

- **ReplicaSet**

설정 되어있는 파드의 개수를 조절해서 항상성 유지합니다.

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

- **Deployment**

레플레카셋에 활용이 되는 pod템플릿 기반을 심어주고, 파드에 대한 항상성 유지하는 역

할을 합니다. 디플로이먼트는 레플리카셋의 상위 오브젝트이기 때문에 디플로이먼트를생성하면 해당 디플로이먼트에 대응하는 레플리카셋도 함께 생성됩니다. 따라서 디플로이먼트를사용하면 포드와 레플리카셋을 직접 생성할 필요가 없습니다.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

- **DaemonSet**

데몬셋(DaemonSets) 은 쿠버네티스의 모든 노드에 동일한 포드를 하나씩 생성하는 오브젝트입니다. 로깅 모니터링 네트워킹 등을 위한 에이전트를 각 노드에 생성해야 할 때 유용하게 사용할 수 있습니다.

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: daemonset-example
spec:
  selector:
    matchLabels:
      name: my-daemonset-example
  template:
    metadata:
      labels:
        name: my-daemonset-example
    spec:
      tolerations: # [3] 마스터 노드에도 포드를 생성
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: daemonset-example
        image: busybox # 테스트를 위해 busybox 이미지 사용
        args: ["tail", "-f", "/dev/null"]
~
```

- **StatefulSet**

상태를 갖는(Stateful) 파드를 관리하기 위한 오브젝트입니다. 스테이트풀셋을 통해 생성되는 파드는 고유한 식별자를 갖습니다. Spec.serviceName이라는 항목이 추가되어 개별 파드에 접근할 수 있도록 합니다.

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: statefulset-example
spec:
  serviceName: statefulset-service
  selector:
    matchLabels:
      name: statefulset-example
    replicas: 3
    template:
      metadata:
        labels:
          name: statefulset-example
      spec:
        containers:
        - name: statefulset-example
          image: sueiin/rr-test:echo-hostname
          ports:
          - containerPort: 80
            name: web
---
apiVersion: v1
kind: Service
metadata:
  name: statefulset-service
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    name: statefulset-example
```

- **Job**

특정 동작을 수행하고 종료해야 하는 작업을 위한 오브젝트입니다. Desired state를 맞추는 파트가 아니고, 특정 task를 수행합니다.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: job-hello-world
spec:
  template:
    spec:
      restartPolicy: Never
      containers:
      - image: busybox
        args: ["sh", "-c", "echo Hello, World && exit 0"]
        name: job-hello-world
```

- **Cronjob**

Job을 주기적으로 실행하는 쿠버네티스 오브젝트입니다. 크론잡을 사용하면 특정 시간 간격으로 job을 반복적으로 실행할 수 있기 때문에 데이터 백업이나 이메일 전송 등의 용도로 사용하기에 적합합니다.

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob-example
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: Never
          containers:
          - name: cronjob-example
            image: busybox
            args: ["sh", "-c", "data" ]
```

2. Discovery & Load Balance 리소스

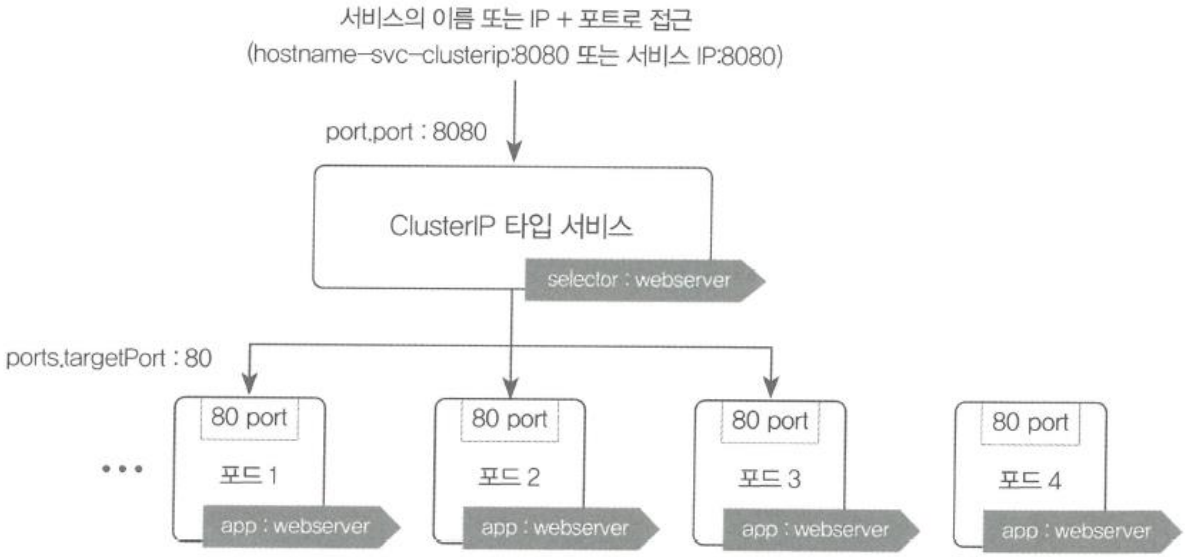
두번째인 Discovery & Load Balance 리소스는 컨테이너의 서비스 디스커버리와 클러스터 내부에서도 액세스 가능한 엔드포인트 등을 제공하는 리소스입니다. 내부적으로 사용되고 있는 것을 제외한 이용자가 직접 조작하는 것은 Service 와 Ingress 인 2 종류의 Discovery & Load Balance 리소스가 있습니다. Service 에는 엔드포인트의 제공 방식이 다른 복수의 타입이 준비되어 있습니다.

[Service]

- **ClusterIP**

쿠버네티스 내부에서만 파드들에 접근할 때 사용합니다. 외부로 파드를 노출하지 않기 때문에 쿠버네티스 클러스터 내부에서만 사용되는 파드에 적합합니다.

```yaml
apiVersion: v1
kind: Service
metadata:
  name:   suejin-svc-clusterip
spec:
  ports:
  - name: web-port
    port: 8080
    targetPort: 80
  selector:
    app: webserver
  type: ClusterIP
```



- **ExternalName**

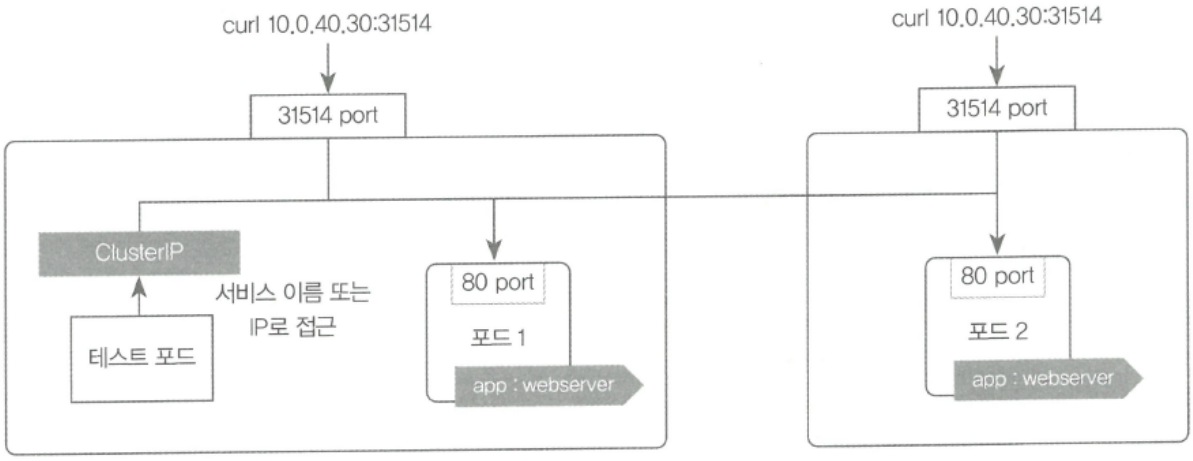
ExternalName 타입을 사용해 서비스를 생성하면 서비스가 외부 도메인을 가리키도록 설정할 수 있습니다. 쿠버네티스와 별개로 존재하는 레거시 시스템에 연동해야 하는 상황에서 유용하게 사용할 수 있습니다.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: externalname-svc
spec:
  type: ExternalName
  externalName: suejin.test.com
```

- **NodePort**

NodePort 타입의 서비스는 클러스터 외부에서도 접근할 수 있습니다. 파드에 접근할 수 있는 포트를 클러스터의 모든 노드에 동일하게 개방합니다. 따라서 외부에서 파드에 접근할 수 있는 서비스 타입입니다. 접근할 수 있는 포트는 랜덤으로 정해지지만, 특정 포트로 접근할 수 있도록 설정할 수 있습니다.

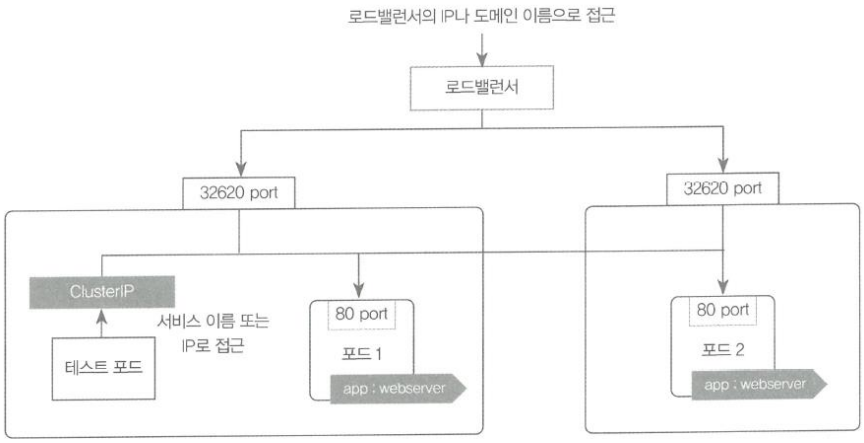
```yaml
apiVersion: v1
kind: Service
metadata:
  name:   suejin-svc-nodeport
spec:
  ports:
  - name: web-port
    port: 8080
    targetPort: 80
  selector:
    app: webserver
  type: NodePort
```



- **LoadBalancer**

LoadBalancer타입의 서비스는 서비스 생성과 동시에 로드 밸런서를 새롭게 생성해 파드와 연결합니다. LoadBalancer 타입의 서비스는 로드 밸런서를 동적으로 생성하는 기능을 제공하는 환경에서만 사용할 수 있습니다. 일반적으로 AWS. GCP 등과 같은 클라우드 플랫폼 환경에서만 LoadBalancer 타입을 사용할 수 있으며 가상 머신이나 온프레미스 환경에서는 사용하기 어려울 수 있습니다.

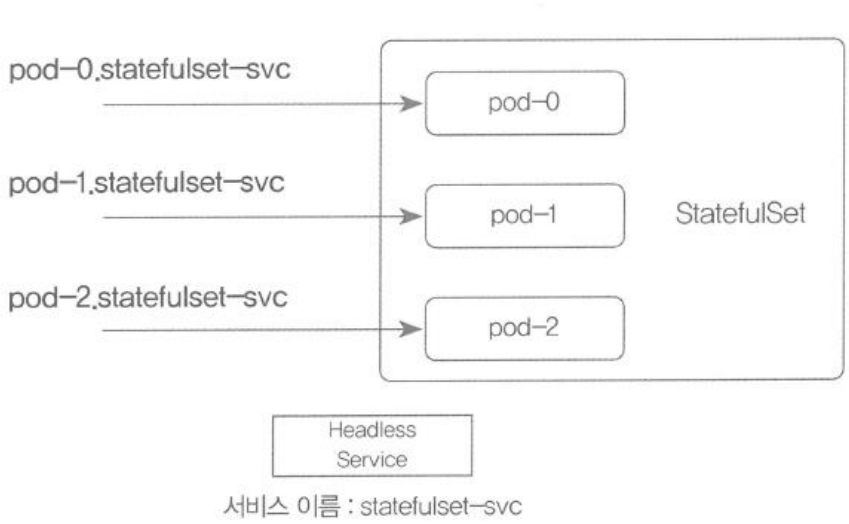
```yaml
apiVersion: v1
kind: Service
metadata:
  name:   suejin-svc-lb
spec:
  ports:
  - name: web-port
    port: 80
    targetPort: 80
  selector:
    app: webserver
  type: LoadBalancer
```



- **Headless**

서비스의 이름으로 파드의 접근 위치를 알아내기 위해서 사용되며, 서비스의 이름과 파드의 이름을 통해서 파드에 직접 접근할 수 있습니다. clusterIP 항목에서 None 을 되있는 부분이 헤드리스 서비스라는 것을 의미합니다.

로드밸런스 타입으로 생성했는데 externalIP 가 할당되지 않으면 사용되는 서비스이기도 합니다.

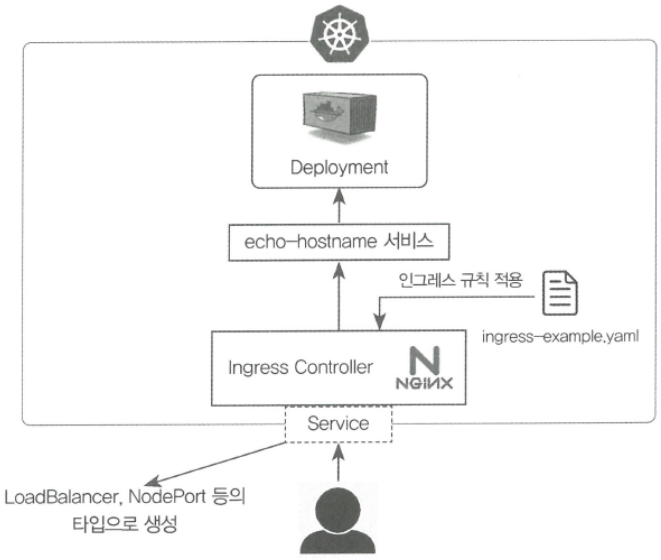


파드의 이름과 헤드리스 서비스를 통한 스테이트풀셋 파드로의 접근 그림입니다.

[Ingress]

인그래스(Ingress)트래픽은 외부에서 서버로 유입되는 트래픽을 의미합니다. 외부 요청을 어떻게 처리할 것인지 네트워크 7 계층 레벨에서 정의하는 쿠버네티스 오브젝트입니다.

- **외부요청의 라우팅:** /apple, /apple/red 등과 같이 특정 경로로 들어온 요청을 어떠한 서비스로 전달할지 정의하는 라우팅 규칙을 설정할 수 있습니다.
- **가상 호스트 기반의 요청 처리:** 같은 IP 에 대해 다른 도메인 이름으로 요청이 도착했을 때, 어떻게 처리할 것인지 정의할 수 있습니다.
- **SSL/TLS 보안 연결 처리:** 여러 개의 서비스로 요청을 라우팅할 때, 보안 연결을 위한 인증서를 쉽게 적용할 수 있습니다.



쿠버네티스가 제공하는 인그레스 오브젝트를 사용하면 하나의 URL 엔드포인트를 생성해 클라이언트가 해당 URL 로 접근하게 되며 라우팅 정의나 보안 연결등과 같은 세부설정을 서비스와 디플로이먼트가 아닌 인그레스에 의해 수행됩니다. 외부 요청에 대한 처리 규칙을 쿠버네티스 자체의 기능으로 편리하게 관리할 수 있습니다.

Nginx, Kong, GKE 등 여러 개의 인그레스를 사용할 수 있지만 Nginx 인그레스를 생성해 보겠습니다.

- **Ingress controller 생성**

실제 외부 요청을 받아들이는 역할을 하고, 컨트롤러 서버가 인그레스 규칙을 로드해서 사용합니다.

$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.0.0/deploy/static/provider/baremetal/deploy.yaml

- **Nginx 인그레스 컨트롤러를 외부로 노출하기 위한 서비스를 생성합니다.**

운영환경으로 AWS 등의 클라우드 환경을 사용하고 있다면 LoadBalancer 타입의 서비스를 생성함으로써 Nginx 인그레스 컨트롤러에 DNS 이름을 할당해 사용합니다.

```yaml
kind: Service
apiVersion: v1
metadata:
  name: lngress-nglnx
  namespace: lngress-nglnx
spec:
  type: loadBalancer
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
  ports:
  - name: http
    port: 80
    targetPort: http
  - name: https
    port: 443
    targetPort: https
```

- **Ingress 생성**

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host:    suejin.example.com # [1]
    http:
     paths:
     - path: /echo-hostname # [2]
       backend:
         serviceName: hostname-service # [3]
         servicePort: 80
```

[1] host: 해당 도메인 이름으로 접근하는 요청에 대해서 처리 규칙을 적용합니다. 여러 개의 host 를 정의해 사용할 수도 있습니다.

[2] Path: 해당 경로에 들어온 요청을 어느 서비스로 전달할 것인지 정의합니다. 여러 개의 path 를 정의해 경로를 처리할 수 있습니다.

[3] serviceName, servicePort: Path 로 들어온 요청이 전달될 서비스와 포트입니다.

- **Nginx 인그레스 컨트롤러로 들어온 요청은 인그레스 규칙에 따라 적절한 서비스로 전달됩니다.**

3. Config & Storage 리소스

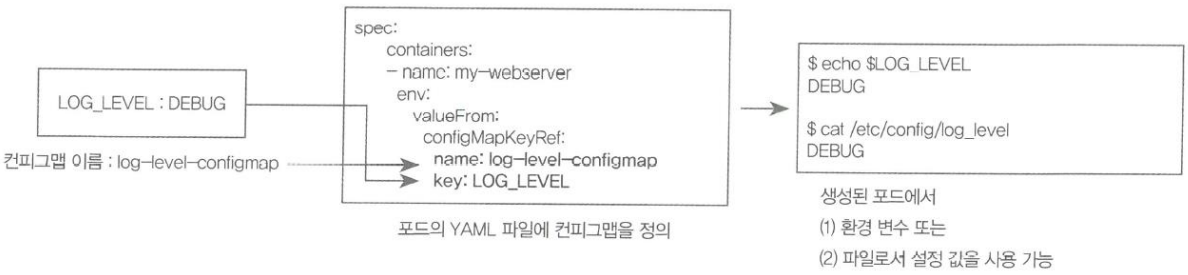
세번째인 Config & Storage 리소스는 설정과 기밀 데이터(Password 등)를 컨테이너에 담거나, Persistent Volume 을 제공하는 리소스입니다. Secret 와 ConfigMap 은 Key-Value 데이터의 구조를 가지며, 보존하고 싶은 데이터가 기밀 데이터인가 일반적인 설정정보인가에 따라 나뉘게 됩니다. PersistentVolumeClaim 은 컨테이너에서 영속 볼륨을 요구할 때에 이용합니다.

- **Secret**

시크릿은 SSH키, 비밀번호 등과 같이 민감한 정보를 저장하기 위한 용도로 사용되며, 네임스페이스에 종속되는 쿠버네티스 오브젝트입니다.

- **ConifgMap**

설정값을 담아 저장할 수 있는 네임스페이스 scope 쿠버네티스 오브젝트입니다.



컨피그맵의 값을 파드의 컨테이너 환경변수로 사용하는 방법이 있고, 컨피그의 값을 파드 내부의 파일로 마운트해서 사용하는 방법이 있습니다.

- **PersistentVolumeClaim**

PersistentVolume 와 파드를 연결하기 위한 리소스입니다. PV는 PersistentVolume으로 물리 디스크를 쿠버네티스 클러스터에 표현한 것입니다.

4. Cluster 리소스

네번째인 Cluster 리소스는 클러스터 자체의 행동을 정의하는 리소스입니다. 보안 설정과 정책, 클러스터의 관리성을 높이는 기능을 위한 리소스들입니다.

- **Node**

쿠버네티스에서 워커 머신을 말한다. 각 노드는 컨트롤 플레인에 의해 관리되며 하나의 노드는 여러 개의 파드를 가질 수 있고, 쿠버니티스 컨트롤 플레인은 클러스터 내 노드를 통해서 파드에 대한 스케줄링을 자동으로 처리한다.

- **Namespace**

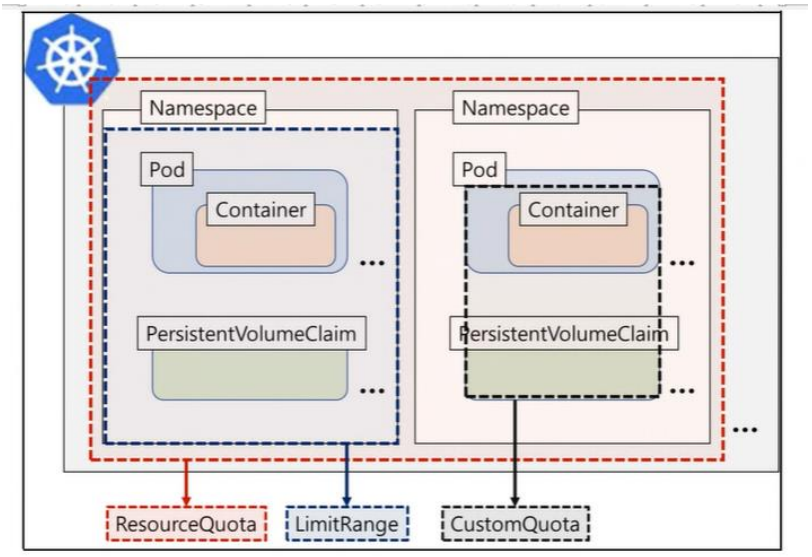
쿠버네티스 클러스터 내의 논리적인 분리 단위입니다. 파드, 레플리카셋, 디플로이먼트, 서비스 등이 묶여 있는 하나의 가상 공간 또는 그룹입니다.

- **PersistentVolume**

PV(PersistentVolume)는 물리 디스크를 쿠버네티스 클러스터에 표현한 것입니다.

- **ResourceQuota**

네임스페이스 별 총 리소스 사용을 제한하는 제약조건입니다. 메모리, cpu, pvc용량, 오브젝트 개수 제한 등의 역할을 합니다.



- **ServiceAccount**

서비스 어카운트는 체계적으로 권한을 관리하기 위한 쿠버네티스 오브젝트입니다. 예를 들어 한 명의 사용자나 애플리케이션에 해당합니다. 네임스페이스에 속하는 오브젝트로 줄여서 sa라는 이름으로 사용할 수 있습니다.

- **Role**

부여할 권한이 무엇인지를 나타내는 쿠버네티스 오브젝트입니다. 네임스페이스에 속하는 오브젝트로, 네임스페이스에 속하는 오브젝트 들에 대한 권한을 정의할 때 사용합니다.

- **ClusterRole**

클러서터 단위의 오브젝트 들에 대한 권한을 정의할 때 사용합니다. 네임스페이스에 속하지 않는 오브젝트 뿐 아니라 클러스터 전반에 걸친 기능을 사용하기 위해서 클러스터 롤을 정의할 수 있습니다. 여러 네임 스페이스에서 반복적으로 사용되는 권한을 클러스터 롤로 만들어 재사용하는 것도 가능 합니다.

- **RoleBinding**

롤바인딩에서는 어떠한 대상을 어떠한 롤에 연결할 것인지 정의합니다.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: service-reader-rolebinding
  namespace: default
subjects:
- kind: ServiceAccount # 권한을 부여할 대상이 ServiceAccount 입니다.
  name: sueiin    #sujin 이라는 이름의 서비스 어카운트에 권한을 부여합니다.
  namespace: default
roleRef:
  kind: Role # Role 어 | 정의된 권한을 부여합니다.
  name: service-reader # service-reader라는 이름으I Role를 대상 (subjects) 어 | 연결합니다.
  apiGroup: rbac.authorization.k8s.io
```

- **ClusterRoleBinding**

클러스터 롤을 특정 대상에게 연결하는 것을 클러스터 롤 바인딩이라고 합니다. 클러스터 롤과 특정 대상을 연결하는 중간 매개체 역할을 합니다. 롤 바인딩과 유사한 역할을 하지만, 클러스터 롤을 위해서 사용할 수 있다는 점이 다릅니다.

- **NetworkPolicy**

IP 주소 또는 포트 수준(OSI 계층 3 또는 4)에서 트래픽 흐름을 제어하려는 경우, 클러스터의 특정 애플리케이션에 대해 쿠버네티스 네트워크폴리시를 사용합니다. 일종의 Pod용 방화벽입니다. 특정 IP나 포트로 부터만 트래픽이 들어오게 하거나 반대로, 특정 IP나 포트로만 트래픽을 내보내게 할 수 있는 등의 설정이 가능합니다.

5. Metadata 리소스

마지막으로 다섯번째인 Metadata 리소스는 클러스터 안에서 다른 리소스의 동작을 제어하기 위한 리소스입니다. 예를 들면 Pod를 오토 스케일링을 시키기 위해 이용되는 HorizontalPodAutoScaler는 Deployment 리소스를 조작하여 복제 수를 변경하는 것으로 오토 스케일링을 실현하고 있습니다.

- **LimitRange**

네임스페이스에서 리소스 할당(파드 or 컨테이너)를 제한하는 정책입니다. Pod와 컨테이너의 최소, 최대 컴퓨팅 리스소 사용량 지정, pvc 최소, 최대 용량 지정, 리밋과 리퀘스트의 할당 비율 지정 등이 있습니다. 리밋은 제한치로 설정 값을 넘으면 안 되고, 리퀘스트는 스케줄러가 참조하는 값으로 리퀘스트 값을 넘게 되면 스케줄링 할 때 프로세스의 제거 순위가 높아지는 등이 수행됩니다.

- **HorizontalPodAutoScalar**

쿠버네티스 워크로드 리소스 (디플로이먼트, 스테이풀셋)을 자동으로 업데이트 하며 워크로드의 크기를 수요에 맞게 자동으로 스케일링 하는 역할을 합니다. Horizontal scaling 은 부하 증가에 따라 파드를 더 배포하는 것이고, Vertical scaling 은 실행되고 있는 파드에 cpu, memory를 더 할당하는 것입니다.

- **PodDisruptionBudget**

파드의 수가 replica의 수를 유지하지 못하고 줄어드는 경우가 있는데, 예상 가능한 상황(업그레이드 등의 이슈)에서 Pod가 없어지는 것을 Voluntary disruptions 라고 하고, 커널 패닉이나 애플리케이션 크래쉬 등의 예상치 못한 상황에서 Pod가 없어지는 것은 Involuntary disruptions라고 한다. 인위적인 노드 다운 등과 같이 Voluntary disruptions 상황에서도 항상 최소한의 Pod 수를 유지하도록 해주는 것이 PodDisruptionBudget(PDB)이다. PDB를 설정하면 관리자가 Pod수를 일정 개수를 유지하지 못하면 노드 다운이나 오토 스케일러에 의한 스케일 다운 등을 막고, Pod 수를 일정 수준으로 유지할 수 있을 때 다시 그 동작을 하도록 한다.

```yaml
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: zk-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: zookeeper
```

- **CustomResourceDefinition**

커스텀리소스(사용자정의리소스)를 정의하는 오브젝트입니다. 커스텀 리소스를 어떻게 사용할 것인지에 대한 쿠버네티스에 등록하는 선언적인 리소스입니다.

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: suejin.k106.com # 1. CRD의 이름
spec:
  group: k106.com # 2. 커스텀 리소스의 API 그룹
  version: v1alpha1 #  커스텀 리소스의 API버전
  scope : Namespaced # 커스텀 리소스가 네임스페이스에 속하는지 여부
  names:
    plural: suejin   # 3. 커스텀 리소스의 이름(복수형)
    singular: suejin #    커스텀 리소스의 이름(단수형)
    kind: Suejin     #    YAML파일 등에서 사용될 커스텀 리소스의 Kind
    shortNames: ["ac"]  # 커스텀 리소스의 줄임말
  validation:
    openAPIV3Schema:    # 4. 커스텀 리소스의 데이터를 정의
    required: ["spec"]  # 커스텀 리소스에는 반드시 "spec" 01 존재해야 함.
    properties:         # 커스텀 리소스에 저장될 데이터 형식을 정의
      spec:
        required: ["myvalue"]
        properties:
          myvalue:
            type: "string ll"
            minimum: 1
```

Reference

1. KubernetesResources – https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
2. KubernetesResources - https://thecodingmachine.tistory.com/9
3. HorizontalPodAutoscaling - https://kubernetes.io/ko/docs/tasks/run-application/horizontal-pod-autoscale/
4. PodDisruptionBudget - https://kubernetes.io/docs/tasks/run-application/configure-pdb/#specifying-a-poddisruptionbudget

5. IngressController - https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/