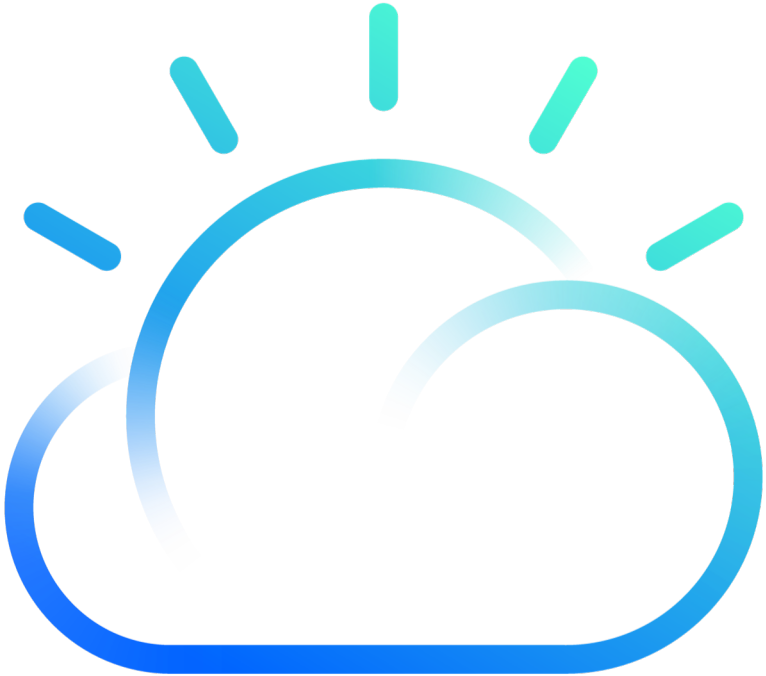
IBM **Cloud** private

an innovation accelerator



Micro segmentation

Jérôme Tarte
IBM Cloud Adoption Leader
jerome.tarte@fr.ibm.com

IBM **Cloud**

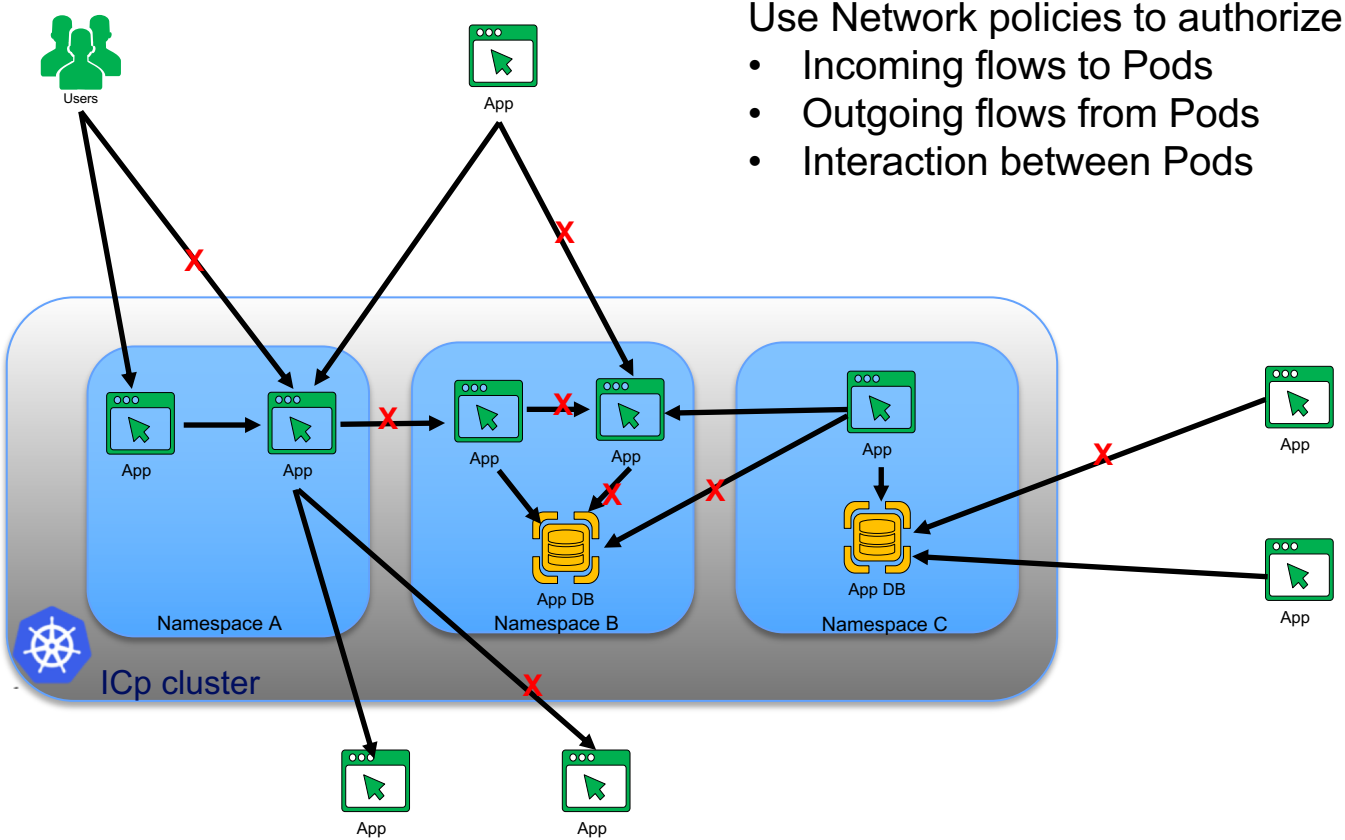
Calico

A new approach to virtual networking and network security for containers, VMs, and bare metal services, that provides a rich set of security enforcement capabilities running on top of a highly scalable and efficient virtual network

- The calico/node Docker container runs on the Kubernetes master and each Kubernetes node in the cluster
- The calico-cni plug-in integrates directly with the Kubernetes kubelet process on each node to discover which pods have been created, and adds them to Calico networking
- The calico/kube-policy-controller container runs as a pod on top of Kubernetes and implements the NetworkPolicy API
- Calico uses the Border Gateway Protocol (BGP) to build routing tables that facilitate communication among agent nodes

PROJECT CALICO

Network micro segmentation



Use Network policies to authorize or not :

- Incoming flows to Pods
- Outgoing flows from Pods
- Interaction between Pods

Kubernetes Network policies

Network policies are implemented by the network plugin

– IBM Cloud private uses Calico as network plugin.

By default, pods are non-isolated

– they accept traffic from any source

– Pods become isolated by having a NetworkPolicy that selects them.

The Network policy is deployed at Namespace level

Network policy resources use labels to select pods and define rules which specify what traffic is allowed to the selected pods.

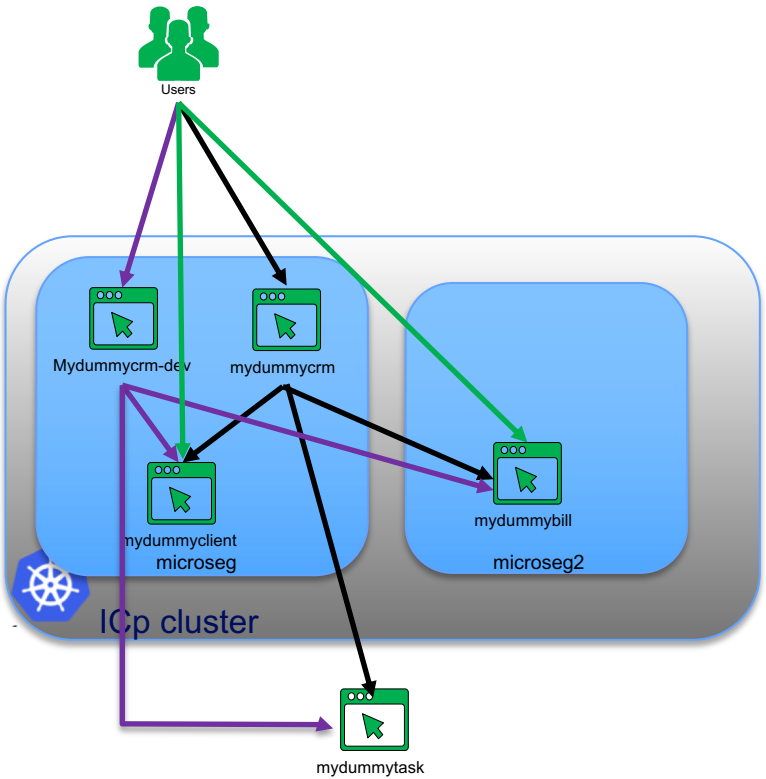
Each NetworkPolicy includes a policyTypes list which may include either Ingress, Egress, or both.

– Ingress rule manages traffic to selected pod

– Egress rule manages traffic from selected pod

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

Micro segmentation in Action : demo architecture



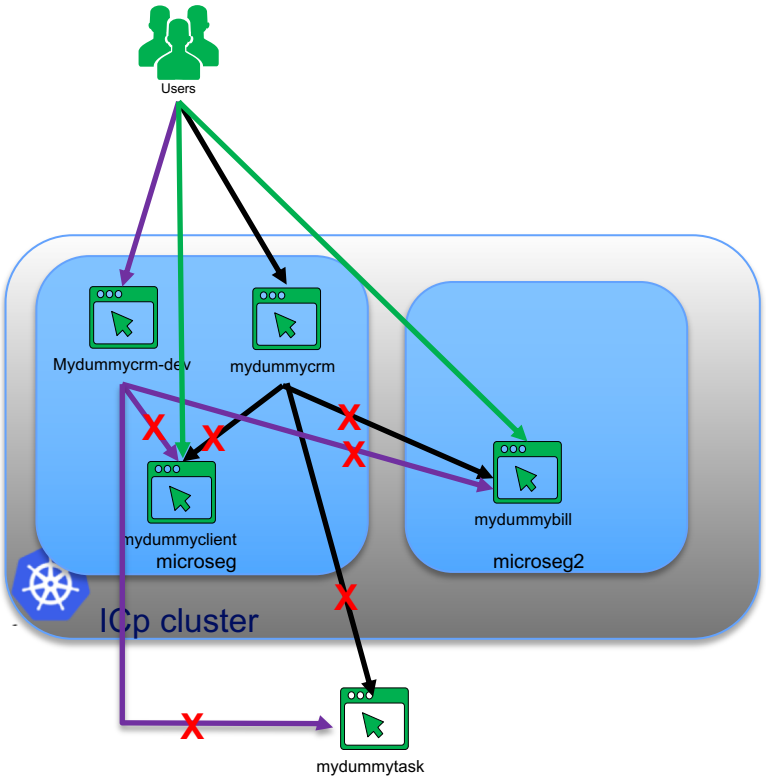
Demo application :

- 1 web UI
- 3 micro services
  - mydummyclient in same namespace than UI
  - mydummybill in a different namespace than UI
  - mydummytask on a server outside of ICp cluster

2 instances of UI : 1 dev and 1 prod

Purpose od the demo: Show how Network policies could affect communication between application components

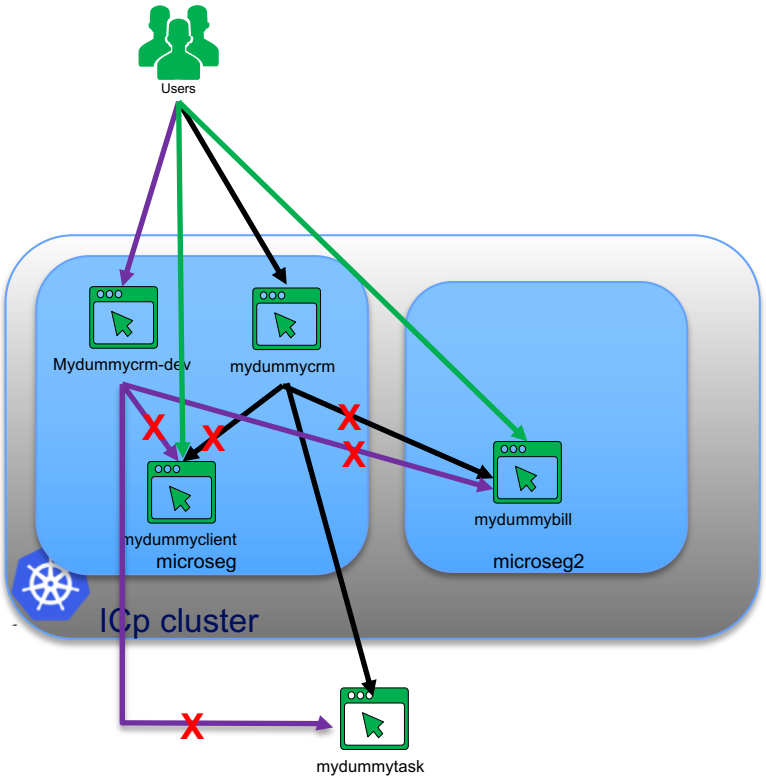
Micro segmentation in Action : Case 1 – Deny all egress traffic



Script : deny_all_egress.yaml

Block all the outgoing communication from pods

Micro segmentation in Action : Case 2 – Deny all egress traffic expect between 1 UI and 1 external microservice



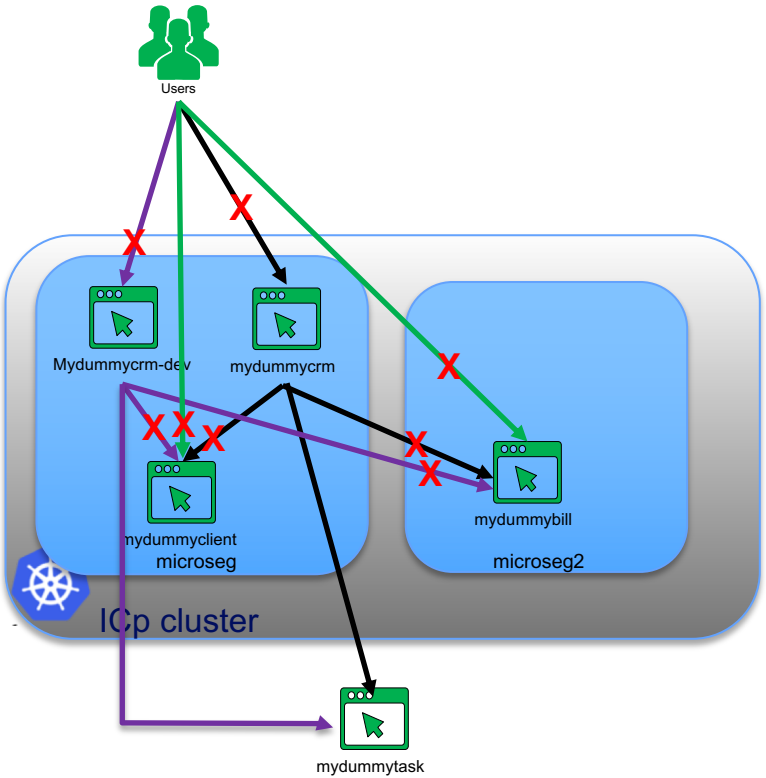
Script : deny_all_egress.yaml

allow_pod_egress.yaml

Block all the outgoing communication from pods

Except for pod mydummycrm that is authorized to contact 172.16.248.152

Micro segmentation in Action : Case 3 – Deny all ingress traffic

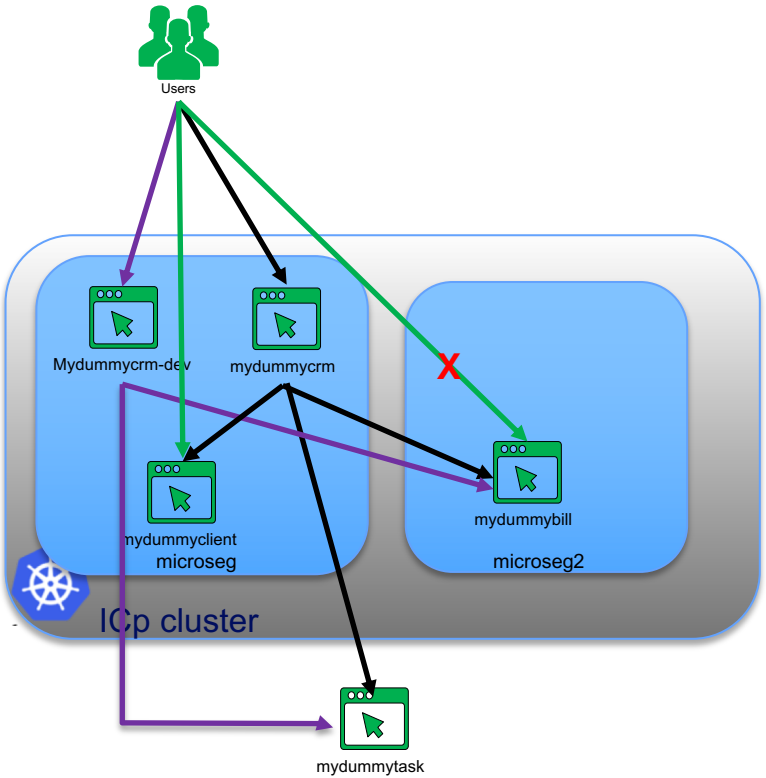


Script: deny-all-ingress.yaml

Block all incoming traffic to pods

Works the two namepsaces -> two network policies

Micro segmentation in Action : Case 4 – Allow ingress traffic from a namespace

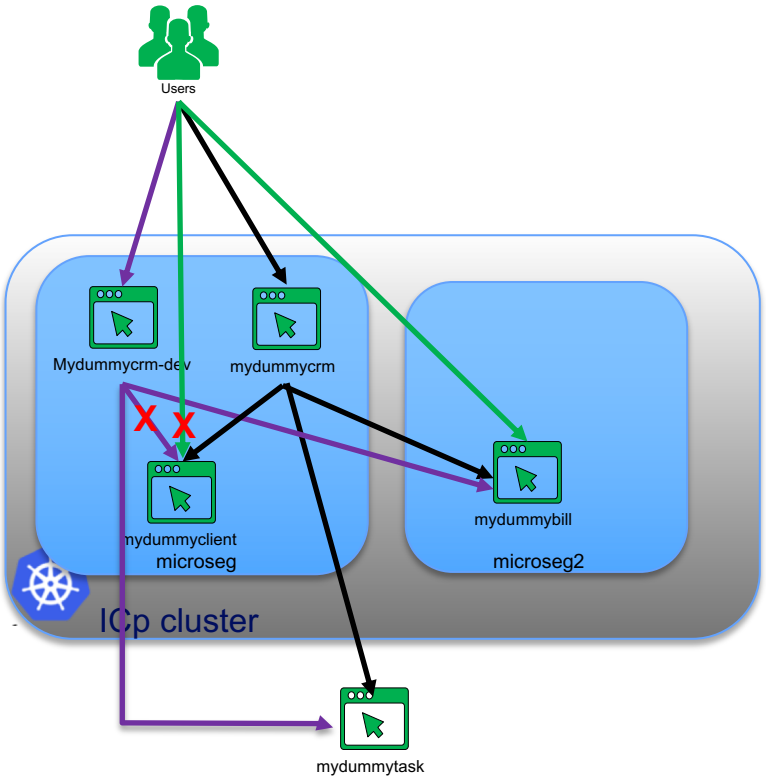


Script : allow_ingress_namespace.yaml

Work on microseg2 namesapce

Block all incoming traffic to pod mydummybill expect from pods of microseg namespace

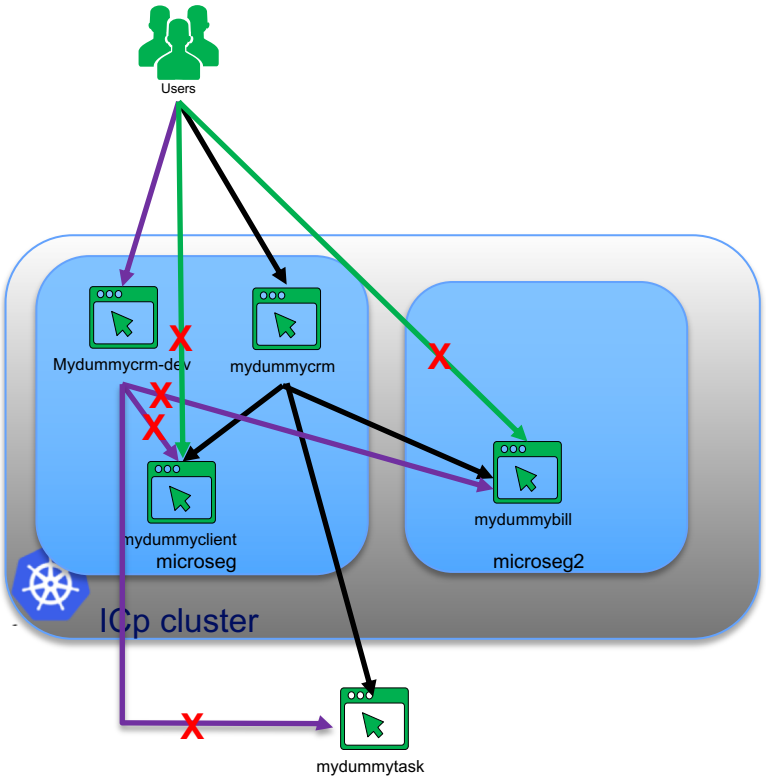
Micro segmentation in Action : Case 5 – Allow ingress traffic from a pod



Script : allow_ingress_pod.yaml

Block all incoming traffic to pod mydummyclient except from pod mydummycrm.

Micro segmentation in Action : Case 6 – full solution



Script : full_solution.yaml

Combination of different network policies to allow only mydummycrm to work.