Aula 16 - Apache Kafka

Disciplina: Arquitetura de Software

Prof. Me. João Paulo Biazotto

UniCesumar

Apache Kafka

- Plataforma distribuída de streaming.
- Desenvolvido originalmente pelo LinkedIn.
- Projetado para alta vazão, tolerância a falhas e escalabilidade.

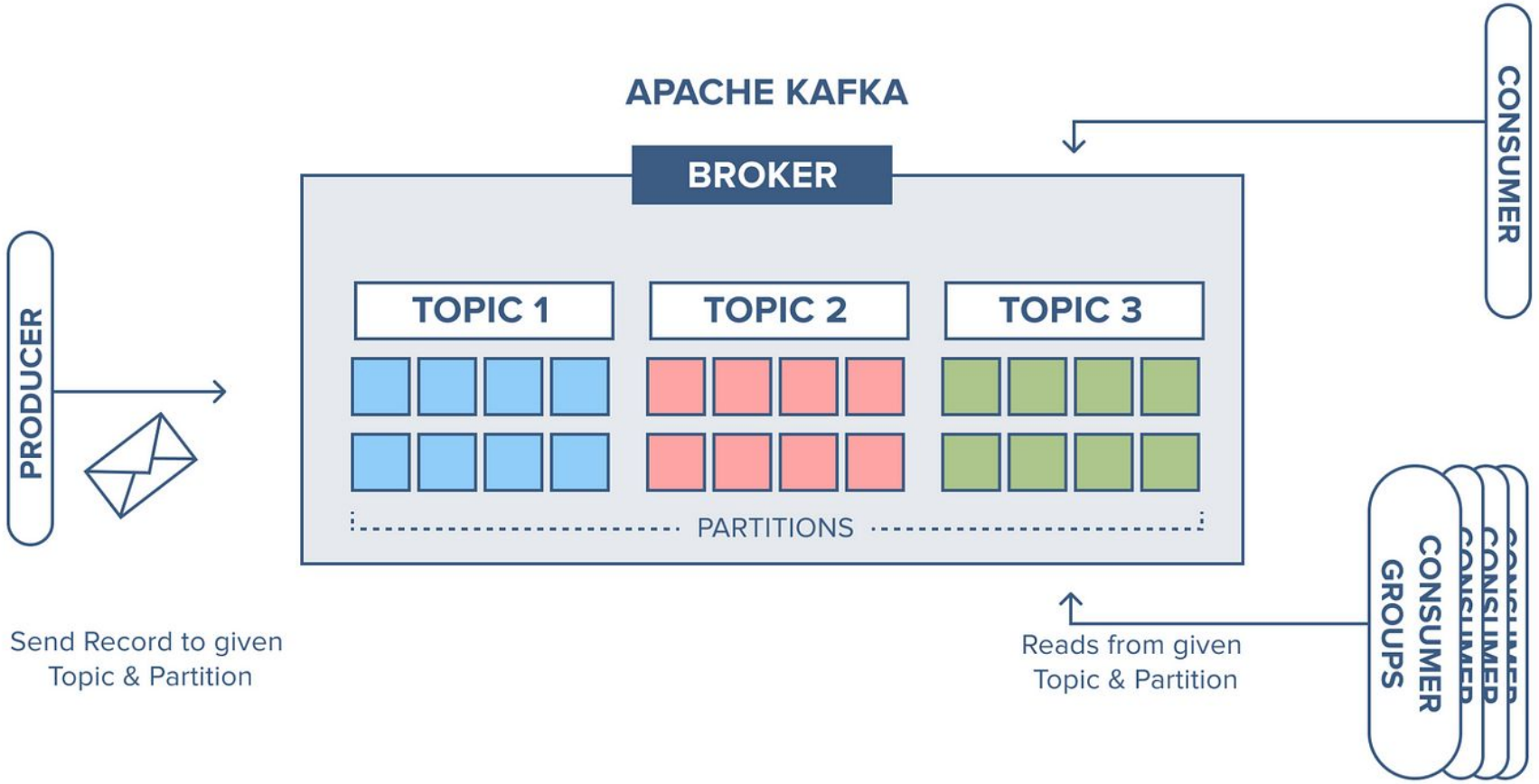
Apache Kafka

- Plataforma distribuída de streaming.
- Desenvolvido originalmente pelo LinkedIn.
- Projetado para alta vazão, tolerância a falhas e escalabilidade.

Apache Kafka

Event streaming (transmissão de eventos) é a prática de capturar dados em tempo real a partir de fontes de eventos como bancos de dados, sensores, dispositivos móveis, serviços em nuvem e aplicações de software, na forma de fluxos de eventos; armazenar esses fluxos de eventos de maneira durável para recuperação posterior; manipular, processar e reagir a esses fluxos de eventos tanto em tempo real quanto de forma retrospectiva; e encaminhar os fluxos de eventos para diferentes tecnologias de destino conforme necessário.

Arquitetura do Kafka



Tópicos

- Um tópico em Kafka é um canal lógico no qual os dados são publicados. Pode-se pensar em tópicos como "categorias" ou "streams" nomeados onde os produtores enviam eventos e os consumidores os leem.

Partições

- Cada tópico pode ser dividido em partições, que são segmentos físicos independentes dos dados.
  - Isso permite paralelismo: múltiplos consumidores podem ler de diferentes partições simultaneamente.
  - Cada partição é ordenada e imutável, e cada mensagem nela tem um identificador único chamado offset.
  - Kafka replica partições em diferentes brokers para garantir tolerância a falhas.

Produção de Dados (Producer)

- Producers são aplicações ou serviços que enviam mensagens para um ou mais tópicos do Kafka.
- Os produtores podem:
  - Enviar mensagens para uma partição específica (com uso de uma chave, por exemplo, ID do usuário).
  - Permitir que o Kafka balanceie automaticamente a partição.

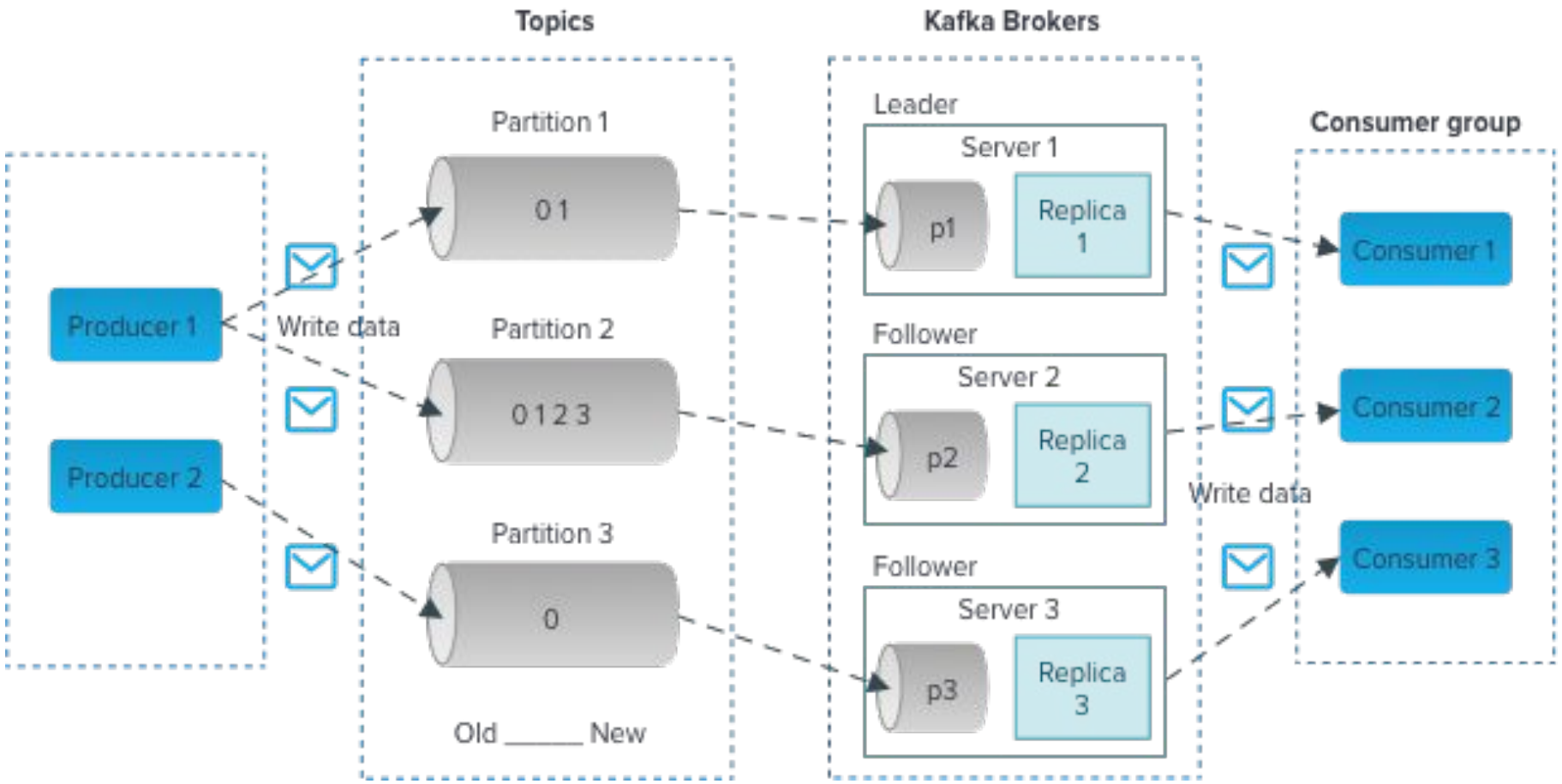
Consumo de Dados (Consumer)

- Consumers leem dados dos tópicos. Cada consumidor lê uma ou mais partições.
- Consumer Groups:
  - Um grupo de consumidores trabalhando juntos para consumir um tópico.
  - Cada partição de um tópico é atribuída a apenas um consumidor do grupo, o que garante paralelismo com exclusividade.
- Rebalanceamento:
  - Quando consumidores entram ou saem do grupo, o Kafka redistribui automaticamente as partições.

Offsets e Retenção

- Offset: É um número sequencial que identifica cada mensagem dentro de uma partição. Os consumidores usam o offset para saber onde continuar a leitura.
- Gerenciamento de offset:
  - Manual ou automático.
  - Armazenado no próprio Kafka (por padrão), ou externamente.
- Retenção:
  - Kafka mantém mensagens por tempo ou tamanho definido (ex.: 7 dias).
  - Não apaga mensagens após leitura — é responsabilidade do consumidor controlar o que já foi processado.

Offsets e Retenção



Resumo Kafka

- Kafka é uma plataforma robusta e distribuída de streaming de eventos.
- Seus conceitos centrais (tópicos, partições, offsets, producers e consumers) permitem alta escalabilidade e desempenho.
- É amplamente usado em pipelines de dados, sistemas reativos, e arquiteturas orientadas a eventos.
- Ferramentas como Kafka Streams e Connect estendem suas funcionalidades para integração e processamento.
- Ideal para sistemas que exigem confiabilidade, tolerância a falhas e grande volume de dados.

Aula 16 - Kubernetes

Disciplina: Arquitetura de Software

Prof. Me. João Paulo Biazotto

UniCesumar

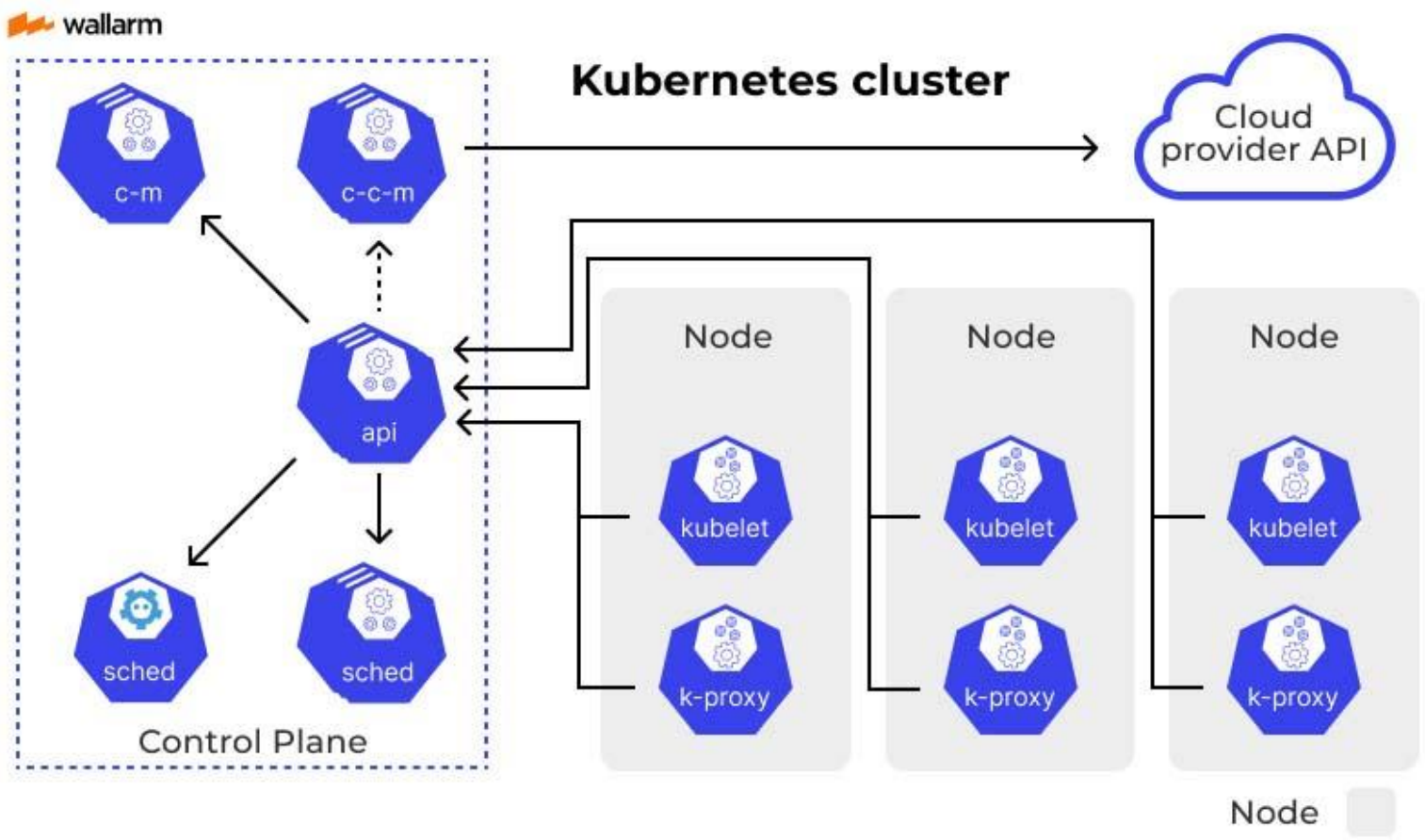
Kubernetes

- Plataforma open source para orquestração de containers.
- Inicialmente desenvolvido pelo Google, hoje mantido pela CNCF.
- Abstrai e automatiza:
  - Implantação (deploy)
  - Escalonamento (scaling)
  - Balanceamento de carga
  - Monitoramento
  - Recuperação de falhas

Conceitos Fundamentais

- Cluster: conjunto de nós (máquinas) controlados centralmente
- Node: instância física ou virtual que executa os pods
- Pod: menor unidade executável; contém um ou mais containers
- Container: aplicação isolada com suas dependências
- Namespace: separação lógica de recursos

Conceitos Fundamentais



Conceitos Fundamentais

- Control Plane (Master Node):
  - kube-apiserver: ponto central de entrada via API
  - etcd: banco de dados chave-valor (estado do cluster)
  - kube-scheduler: define onde executar pods
  - controller-manager: gerencia ciclos de vida de recursos

Conceitos Fundamentais

- Worker Nodes:
  - kubelet: agente que gerencia o estado do node
  - kube-proxy: gerenciamento de rede e balanceamento
  - container runtime: ex. Docker, containerd, CRI-O

Pods

- Cada pod encapsula:
  - Um ou mais containers
  - Recursos compartilhados: rede, volumes
- Pode morrer e ser recriado por controllers
- Casos de múltiplos containers no pod:

ReplicaSets e Deployments

- ReplicaSet:
  - Garante que n réplicas de um pod estejam sempre em execução.
  - Substitui o conceito de “scale” do Docker.

- Deployment:
  - Abstração de alto nível que gerencia ReplicaSets.
  - Permite atualizações contínuas (rolling updates) e rollback automático.

Services e Descoberta

- Service: abstração para acessar um ou mais pods de forma estável
- Tipos:
  - ClusterIP: acessível apenas dentro do cluster
  - NodePort: expõe um serviço em portas dos nós
  - LoadBalancer: utiliza balanceador de carga externo (ex. AWS, GCP)
- Service Discovery:
  - DNS interno com mapeamento de nome de serviço para IP

ConfigMaps e Secrets

- ConfigMaps:
  - Armazenam pares chave-valor de configurações não sensíveis

- Secrets:
  - Armazenam informações sensíveis, como senhas e tokens, de forma codificada (base64)
  - Ambos podem ser montados como arquivos ou variáveis de ambiente nos pods

Volumes e Persistência

- Volumes: armazenamento compartilhado entre containers de um pod
- Tipos:
  - emptyDir: temporário (enquanto o pod vive)
  - hostPath: usa sistema de arquivos do node
  - persistentVolume (PV) + persistentVolumeClaim (PVC): persistência desacoplada do ciclo de vida do pod
- Storage Classes permitem provisionamento dinâmico com diferentes backends (ex. EBS, NFS)

Escalonamento e Auto-Scaling

- Horizontal Pod Autoscaler (HPA): Aumenta ou reduz o número de pods com base em CPU, memória ou métricas customizadas
- Vertical Pod Autoscaler (VPA): Ajusta os recursos (CPU/memória) alocados a um pod
- Cluster Autoscaler: Adiciona ou remove nós do cluster baseado na demanda

Monitoramento e Logging

- Logs:
  - Coletados por agentes como Fluentd, Logstash
  - Enviados para sistemas como Elasticsearch, Loki, etc.
- Monitoramento:
  - Prometheus + Grafana: métricas de pods, nodes e aplicações
  - kube-state-metrics: coleta de dados sobre o estado do cluster
- Importância: detectar gargalos, falhas e prever necessidade de escala

Segurança

- RBAC (Role-Based Access Control): Controle de permissões por usuário, grupo ou serviço
- Namespaces: Isolamento de ambientes (dev, staging, prod)
- Service Accounts: Identidade de pods para acesso a recursos internos
- Network Policies: Controla o tráfego entre pods com regras explícitas

Estudo de Caso

- Exemplo prático:
  - Aplicação e-commerce dividida em microserviços: catálogo, pagamento, carrinho
- Cada microserviço:
  - Roda em seu próprio pod com réplica via Deployment
  - Comunica-se por meio de Services internos
  - Configurações via ConfigMap, segredos via Secret
  - Monitorado com Prometheus e logs centralizados
  - Deploy via CI/CD com ferramentas como ArgoCD ou Jenkins + Helm

Resumo Kubernetes

- Kubernetes transforma o modo como aplicações são implantadas e escaladas.
- Conceitos como pods, deployments, services e volumes são fundamentais.
- Escalonamento automático e abstrações ricas facilitam resiliência e performance.
- Kubernetes é essencial em arquiteturas modernas baseadas em containers e microserviços.

DÚVIDAS?

UniCesumar