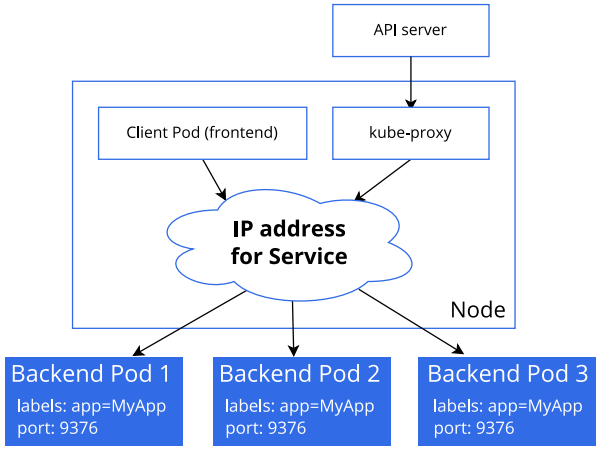
Kubernetes サービスとネットワーキング

クラスター内の通信と外部公開



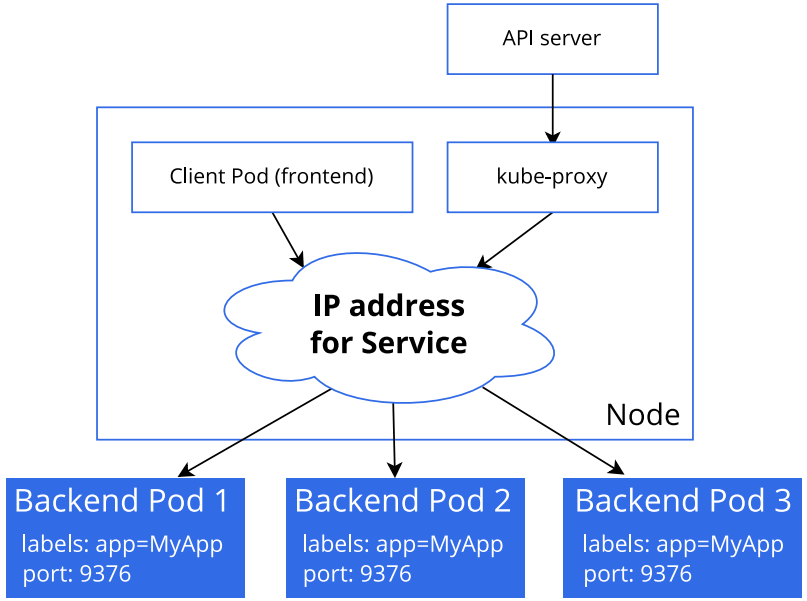
出典: Kubernetes公式ドキュメント - Services, Load Balancing, and Networking

Serviceとは

- Podへの安定したアクセスを提供する抽象化レイヤー
- PodのIPアドレスは動的に変更されるため、直接アクセスは非推奨
- Serviceは固定のIPアドレスとDNS名を提供

> □ **重要**: ServiceはPodの集合に対する論理的なエンドポイントを提供します

Serviceの基本概念



出典: Kubernetes公式ドキュメント - Services, Load Balancing, and Networking

- ServiceはPodの集合に対する論理的なエンドポイント
- kube-proxyがiptablesルールを管理
- クラスター内DNSによる名前解決

Serviceの種類 (1/2)

1. ClusterIP (デフォルト)

- クラスター内部からのみアクセス可能
- 内部サービス間の通信に使用

2. NodePort

- クラスター外からノードのIP:ポートでアクセス可能
- 開発・テスト環境で使用

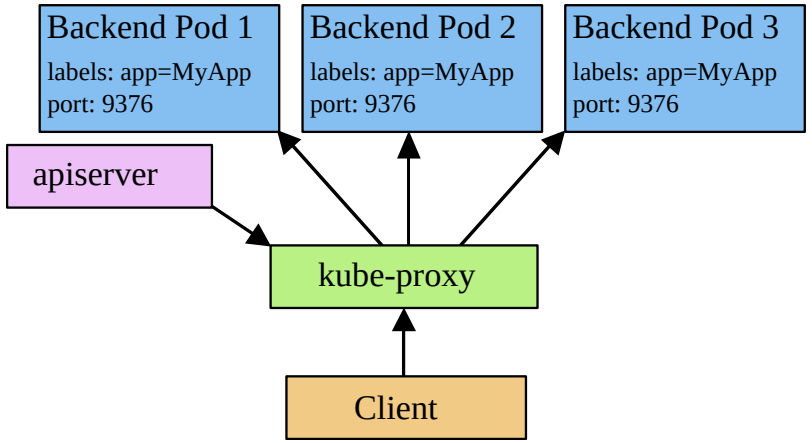
Serviceの種類 (2/2)

3. LoadBalancer

- クラウドプロバイダーのロードバランサーを使用
- 本番環境での外部公開に使用

4. ExternalName

- 外部サービスへのDNSエイリアスを提供



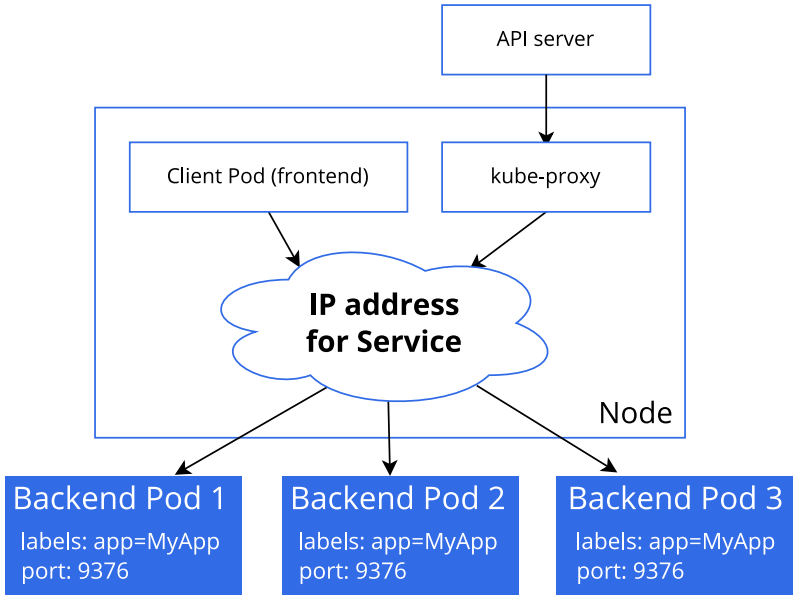
出典: Kubernetes公式ドキュメント - Services, Load Balancing, and Networking

ClusterIP Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: ClusterIP
```

□ **特徴**: - クラスター内部でのみアクセス可能 - 自動的にクラスター内DNSに登録 - 例: `my-service.default.svc.cluster.local`

ClusterIP Serviceの動作



出典: Kubernetes公式ドキュメント - Services, Load Balancing, and Networking

- kube-proxyがiptablesルールを管理
- ラウンドロビンによる負荷分散
- クラスター内DNSによる名前解決

NodePort Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 8080
      nodePort: 30000
```

⚠ **注意点**: - ポート範囲: 30000-32767 - セキュリティ考慮が必要 - 本番環境ではLoadBalancerの使用を推奨

NodePort Serviceの動作

出典: Kubernetes公式ドキュメント - Services, Load Balancing, and Networking

- すべてのノードの指定ポートでアクセス可能
- ノード間での負荷分散
- クラスター外からのアクセスに使用

LoadBalancer Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: LoadBalancer
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 8080
```

□ **利点**: - クラウドプロバイダーのロードバランサーを自動プロビジョニング - 外部からのアクセスを複数ノードに分散 - 本番環境での推奨方式

LoadBalancer Serviceの動作

出典: Kubernetes公式ドキュメント - Services, Load Balancing, and Networking

- クラウドプロバイダーのロードバランサーを使用
- 外部からのアクセスを複数ノードに分散
- 高可用性とスケーラビリティを提供

Ingress (1/2)

- HTTP/HTTPSトラフィックのルーティングを管理
- ホスト名やパスベースのルーティング
- SSL/TLS終端
- ロードバランシング

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service
            port:
```

Ingress (2/2)

出典: Kubernetes公式ドキュメント - Services, Load Balancing, and Networking

- 複数のServiceを単一のエンドポイントで公開
- パスベースのルーティング
- SSL/TLS終端
- ロードバランシング

ネットワークポリシー (1/2)

- Pod間の通信を制御
- 名前空間レベルでの分離
- セキュリティポリシーの実装

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
spec:
  podSelector:
    matchLabels:
      app: my-app
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

ネットワークポリシー (2/2)

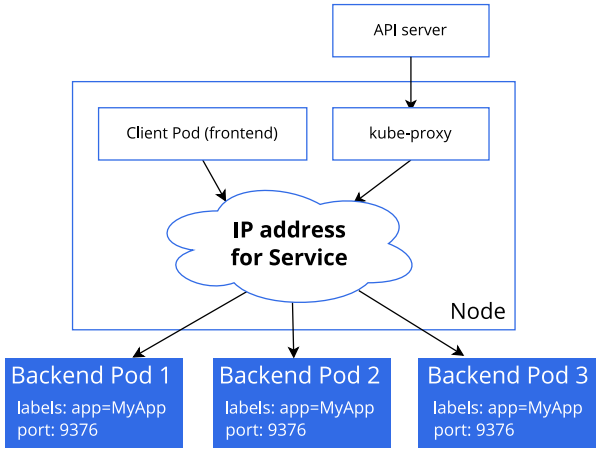
出典: Kubernetes公式ドキュメント - Services, Load Balancing, and Networking

- Pod間の通信を制御
- 名前空間レベルでの分離
- セキュリティポリシーの実装
- マイクロサービス間の通信制御

実践的な使用例 (1/2)

マイクロサービス構成

- フロントエンド → API → データベース
- 各層で適切なServiceタイプを選択
- セキュリティ考慮した通信制御



出典: Kubernetes公式ドキュメント - Services, Load Balancing, and Networking

実践的な使用例 (2/2)

ベストプラクティス

- 最小権限の原則
- 適切なServiceタイプの選択
- ネットワークポリシーの活用
- モニタリングとロギング

 推奨事項: - 本番環境ではLoadBalancer + Ingress - 内部サービスはClusterIP - ネットワークポリシーで通信制限

まとめ

- ServiceはPodへの安定したアクセスを提供
- 用途に応じて適切なServiceタイプを選択
- Ingressによる柔軟なルーティング
- ネットワークポリシーによるセキュリティ制御
- 実運用ではモニタリングと管理が重要

 参考資料: - [Kubernetes公式ドキュメント](https://kubernetes.io/docs/concepts/services-networking/) - [Kubernetes Service Types](https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types) - [Ingress Controllers](https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/)

ご清聴ありがとうございました

- 質問はありますか？
- より詳しい情報は公式ドキュメントを参照してください

□ **次のステップ**: - 実際のクラスターでServiceを作成 - 異なるServiceタイプの動作確認 - Ingressコントローラーの設定 - ネットワークポリシーの実装