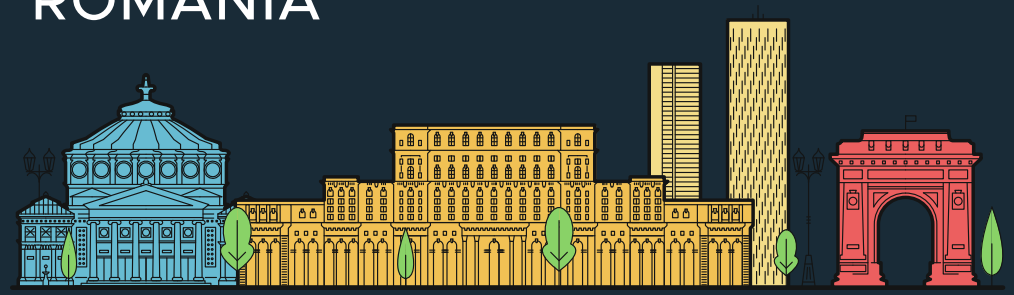
endava

ROMANIA



CLOUD NATIVE DAYS

Workshop:
Amazon EKS Autoscaling.

Claudiu Șonel

Senior DevOps Consultant

sonel.claudiu@endava.com

Claudiu is an experienced Senior Engineer with a demonstrated history of working in the information technology and services industry. Skilled in Linux System Administration, AWS, GCP, Jenkins, Docker, Kubernetes and Infrastructure as Code. Claudiu has successfully designed and managed complex cloud-native infrastructures for diverse organizations.

As a passionate advocate for automation and scalability, Claudiu specializes in leveraging Kubernetes to create resilient, cost-effective , and high-performing systems. Claudiu empowers teams to unlock the full potential of AWS and Kubernetes for modern workloads.

The Problem

Scaling is the backbone of modern cloud infrastructure. In a microservices architecture, each service can independently scale based on its specific needs. Kubernetes Autoscaling ensures that your application remains resilient and responsive by dynamically adjusting resources. Whether it's handling a sudden surge in traffic or optimizing costs during low usage periods, autoscaling provides the flexibility and efficiency that today's applications demand

Agenda

01 Kubernetes

02 AWS EKS

03 EKS Autoscaling

04 Conclusion

Code:

https://github.com/csonel/workshop-cndro-2025

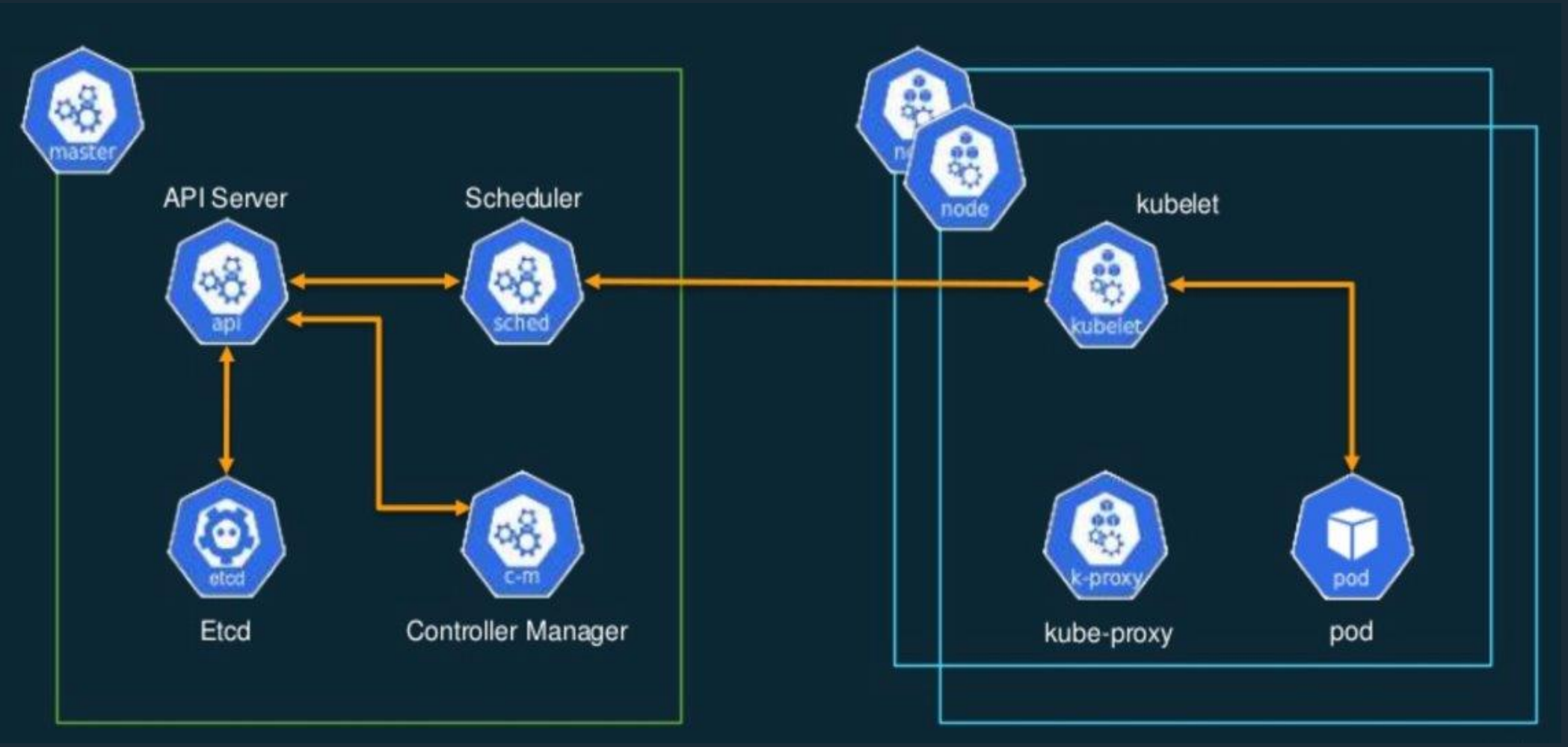
AWS EKS.

Chapter

01

Kubernetes

Kubernetes Cluster Architecture.



Let's talk the same language

Glossary

Kubernetes terms:

- **Pod**
  co-located group of containers that shares an IP, namespace, storage volume.
- **Deployment**
  manages the lifecycle of pods and ensures specified number of replicas are running.
- **Service**
  single, stable name for a set of pods, also acts as a Load Balancer.
- **Namespace**
  a way to divide cluster resources between multiple users (via resource quota).
- **Label**
  used to organize and select group of object.

Chapter

02

Amazon Elastic Kubernetes Service (EKS)

Let's start with Amazon EKS

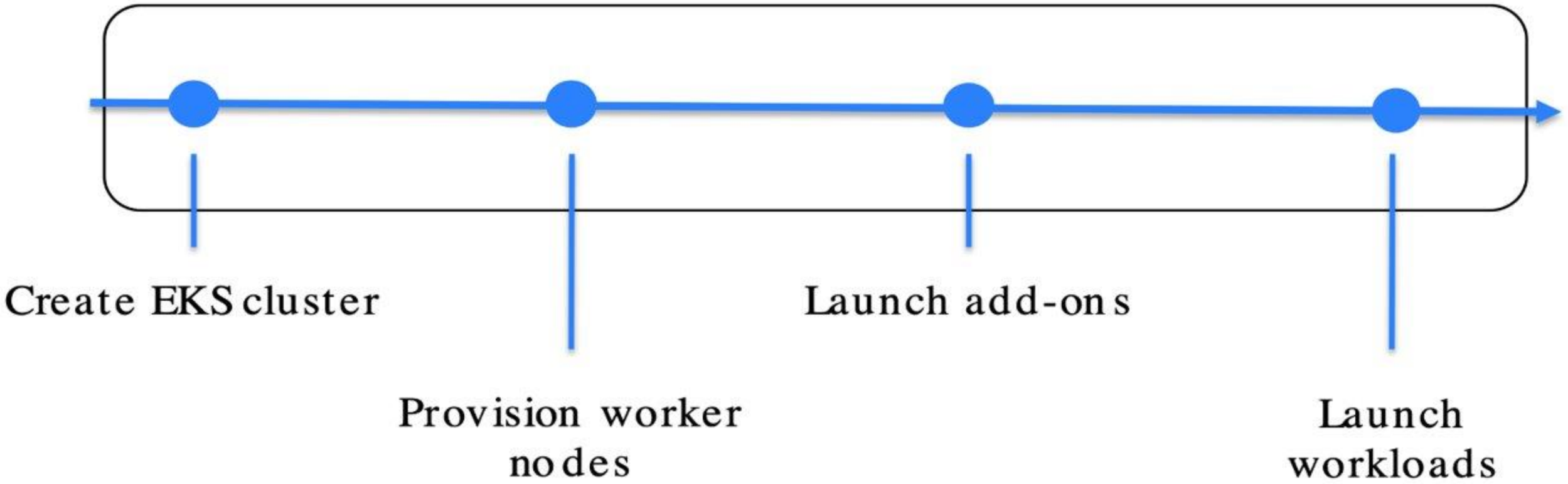
What is AWS EKS?

EKS is a managed service that make it easy for you to run fully compatible Kubernetes on AWS without needing to stand up or maintain your own Kubernetes control plane.

AWS service Integration:

- **Elastic Load Balancing**
- **IAM**
- **Amazon VPC**
- **AWS Fargate**
- **CloudWatch**
- **CloudTrail**
- **...**

EKS Cluster Creation Workflow.

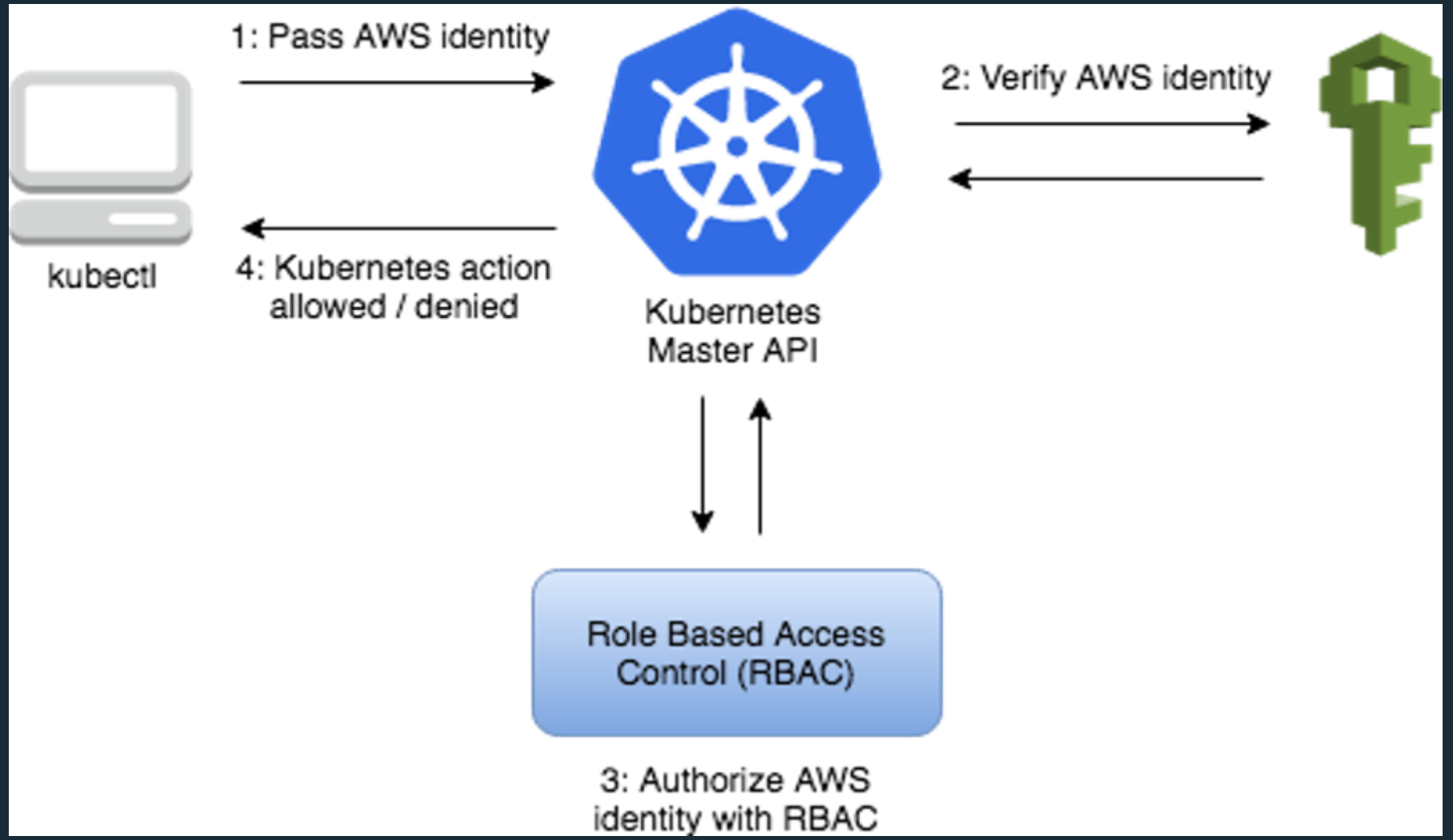


Kubernetes API Server Endpoint Access Control

API server endpoint access
- Public Access
- Private Access

Endpoint Public Access	Endpoint Private Access	Behavior
Enabled	Disabled	• This is the default behavior for new Amazon EKS clusters. • Kubernetes API requests that originate from within your cluster's VPC (such as worker node to control plane communication) leave the VPC but not Amazon's network. • Your cluster API server is accessible from the internet.
Enabled	Enabled	• Kubernetes API requests within your cluster's VPC (such as worker node to control plane communication) use the private VPC endpoint. • Your cluster API server is accessible from the internet.
Disabled	Enabled	• All traffic to your cluster API server must come from within your cluster's VPC. • There is no public access to your API server from the internet. Any `kubectl` commands must come from within the VPC as well. For connectivity options, see Accessing the API Server from within the VPC.

EKS Authentication.



aws-auth ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn: arn:aws:iam::123456789012:role/eks-worker-node-role
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
  mapUsers: |
    - userarn: arn:aws:iam::123456789012:user/eks-admin
      username: admin
      groups:
        - system:masters
    - userarn: arn:aws:iam::123456789012:user/john
      username: john
      groups:
        - pod-admin # k8s RBAC group
```

Chapter

03

Autoscaling Workshop
Let’s get our hands dirty

Chapter

04

Conclusion

Primary function

Cluster Autoscaler

Scales the number of nodes in a cluster based on pod resource requests and limits.

Karpenter

Provides more intelligent and flexible scaling decisions, optimizing resource utilization and reducing latency.

Scaling Triggers

Cluster Autoscaler

Pod resource requests and limits.

Karpenter

Real-time workload demands and custom policies.

Flexibility

Cluster Autoscaler

Limited to scaling nodes based on pod requirements.

Karpenter

Highly flexible, can scale based on various metrics and policies.

Latency

Cluster Autoscaler

Moderate - scales based on predefined thresholds.

Karpenter

Low - provides faster scaling decisions to meet real-time demands.

Configuration Complexity

Cluster Autoscaler

Moderate - requires configuration of resource requests and limits for pods.

Karpenter

High - offers more advanced features and requires deeper understanding of scaling policies.

Resource Utilization

Cluster Autoscaler

Good - ensures pods have the necessary resources.

Karpenter

Excellent - optimizes resource utilization across the cluster.

Cost Efficiency

Cluster Autoscaler

Good - helps manage costs by scaling nodes as needed.

Karpenter

Excellent - optimizes costs by making intelligent scaling decisions.

Use Case Suitability

Cluster Autoscaler

Best for standard scaling needs in Kubernetes clusters.

Karpenter

Ideal for complex, dynamic workloads requiring advanced scaling capabilities.

Thank you!