

Kubernetes Cheatsheet

TABLE OF CONTENTS

Preface	1
Introduction	1
Kubernetes Basics	1
Quick Reference Table	1
Architecture Overview	1
Essential Commands	1
Pods and Containers	2
Cheat sheet table: Common commands for creating, managing, and troubleshooting pods	2
Best practices for effective containerization within the Kubernetes environment	2
Code snippets for creating multi-container pods	3
Deployments and ReplicaSets	3
Visual guide: Lifecycle of Deployments and how ReplicaSets ensure high availability	4
Cheat sheet for scaling applications with Deployments and managing rolling updates	4
Code examples for creating and updating ReplicaSets	4
Services and Networking	5
Graphical representation: Kubernetes networking components and their interactions	5
Cheat sheet for creating and managing Kubernetes services	5
YAML examples for defining Services and exposing applications	6
Configuration and Secrets	6
Key configurations in Kubernetes and practical use cases	6
Secure practices for managing and consuming secrets	7
Code snippets for using ConfigMaps and Secrets in Kubernetes pods	7
Monitoring and Troubleshooting	7
Cheat sheet for troubleshooting common issues in Kubernetes clusters	8
Visual representation of Kubernetes dashboard components	8
Advanced Topics	8
Cheat sheet for Helm charts and efficient package management in Kubernetes	8
Code examples for setting up Persistent Volumes and Persistent Volume Claims	8
Additional Commands	9
For Node Management	9
For Namespace Management	9
Service Commands	10
Deployment Commands	10
Resources	10

PREFACE

Welcome to the Kubernetes Cheatsheet! In this cheatsheet, you'll find essential commands, concepts, and configurations to help you navigate and leverage Kubernetes effectively. Whether you're a beginner exploring containerization or a seasoned DevOps professional, this cheatsheet aims to provide quick references and reminders for common Kubernetes tasks, enabling you to streamline your workflow and harness the full potential of containerized environments.

INTRODUCTION

Kubernetes, commonly referred to as K8s, is a widely adopted container orchestration platform that simplifies the deployment, scaling, and management of containerized applications.

In the traditional model, deploying applications across different environments posed challenges due to variations in infrastructure and configurations. Kubernetes addresses these challenges by providing a unified platform that abstracts away the complexities of the underlying infrastructure. It allows developers to deploy and manage applications consistently across various environments, fostering portability and scalability.

Kubernetes, like Docker, emphasizes the use of container technology. However, while Docker focuses on packaging and distribution, Kubernetes takes containerization to the next level by providing tools for automating deployment, scaling, and operations.

A Kubernetes cheat sheet is a must-have, providing quick access to essential commands for seamless deployment management. Let's look at it as your go-to guide in the fast-paced Kubernetes environment, enhancing productivity, minimizing errors, and ensuring efficient navigation through complex tasks.

KUBERNETES BASICS

QUICK REFERENCE TABLE

The quick reference table below serves as a compass for navigating the terminology crucial to mastering Kubernetes.

Term	Meaning
Pod	Smallest deployable unit in Kubernetes.
Node	A physical or virtual machine in a cluster.
Deployment	Describes the desired state for a set of pods.
Service	Defines a set of pods and a policy for accessing them.
Labels	Key-value pairs attached to objects, allowing for flexible categorization and selection. Used for identifying and grouping related resources.
Master	The control plane in Kubernetes, managing the overall state of the cluster. Components include the API server, etcd, controller manager, and scheduler.

ARCHITECTURE OVERVIEW

Kubernetes operates on a client-server architecture, comprising a control plane and a set of nodes where containers run.

Figure 1. Architecture Overview

ESSENTIAL COMMANDS

With a foundational understanding of terms and architecture, let's equip ourselves with essential commands for managing Kubernetes clusters.

These commands are crucial for understanding the status, configuration, and available resources within a Kubernetes cluster. They provide a quick

overview of the cluster's information, version, configuration, available API resources, and versions.

Command	Description
<code>kubectl cluster-info</code>	Display endpoint information about the master and services in the cluster.
<code>kubectl version</code>	Display the Kubernetes version running on the client and server.
<code>kubectl config view</code>	Get the configuration of the cluster.
<code>kubectl api-resources</code>	List the API resources that are available.
<code>kubectl api-versions</code>	List the API versions that are available.
<code>kubectl get all --all-namespaces</code>	List everything in all namespaces.

PODS AND CONTAINERS

A Pod is the smallest deployable unit in Kubernetes, representing a single instance of a running process in a cluster. Pods are the basic building blocks and can contain one or more containers that share the same network namespace, allowing them to communicate with each other using localhost.

A Container is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools. Containers provide a consistent and isolated environment for applications, ensuring that they run consistently across different environments.

CHEAT SHEET TABLE: COMMON COMMANDS FOR CREATING, MANAGING, AND TROUBLESHOOTING PODS

Command	Description
<code>kubectl apply -f <pod_configuration></code>	Apply the configuration specified in a YAML file to create or update a pod.

Command	Description
<code>kubectl get pods</code>	List all pods in the default namespace.
<code>kubectl get pods --all-namespaces</code>	List all pods in all namespaces.
<code>kubectl describe pod <pod_name></code>	Display detailed information about a specific pod.
<code>kubectl logs <pod_name></code>	View the logs of a specific pod.
<code>kubectl exec -it <pod_name> /bin/bash</code>	Open an interactive shell inside a running container.
<code>kubectl delete pod <pod_name></code>	Delete a specific pod.
<code>kubectl apply -f <pod_configuration></code>	Apply the configuration specified in a YAML file to create or update a pod.
<code>kubectl port-forward <pod_name> <local_port>:<pod_port></code>	Forward ports from a pod to your local machine.

BEST PRACTICES FOR EFFECTIVE CONTAINERIZATION WITHIN THE KUBERNETES ENVIRONMENT

Effective containerization is crucial for smooth application deployment and management in a Kubernetes environment. Here are key best practices to optimize your containerized applications:

- ¥ Adhere to the Single Responsibility Principle, ensuring each container has a specific and well-defined role. This promotes modularity and simplifies maintenance.
- ¥ Minimize the size of your container images by reducing unnecessary layers and dependencies. Smaller images improve deployment speed and reduce resource consumption.
- ¥ Leverage environment variables to configure your application. This enhances flexibility, making it easier to adjust settings without modifying the container image.
- ¥ Implement health probes within your application to enable Kubernetes to assess its health. This helps in automated healing and

ensures reliable application performance.

¥ Set resource limits, such as CPU and memory, to prevent containers from consuming excessive resources. This ensures fair resource allocation and prevents one misbehaving container from impacting others on the same node.

CODE SNIPPETS FOR CREATING MULTI-CONTAINER PODS

Using multi-container pods in Kubernetes allows for enhanced collaboration and resource sharing between containers within the same pod. Here are code snippets demonstrating different multi-container patterns.

Sidecar Pattern

The sidecar container runs alongside the main application container, providing additional functionalities without altering the core application.

```
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-pod
spec:
  containers:
  - name: main-container
    image: main-image:latest
    ports:
    - containerPort: 80
  - name: sidecar-container
    image: sidecar-image:latest
```

Ambassador Pattern

In this pattern, the ambassador container acts as an intermediary, handling communication and network-related tasks on behalf of the main application.

```
apiVersion: v1
kind: Pod
metadata:
  name: ambassador-pod
spec:
  containers:
```

```
  - name: main-container
    image: main-image:latest
    ports:
    - containerPort: 80
  - name: ambassador-container
    image: ambassador-image:latest
    ports:
    - containerPort: 8080
```

Adapter Pattern

The adapter container transforms or adapts data before it reaches the main application, providing a seamless integration point.

```
apiVersion: v1
kind: Pod
metadata:
  name: adapter-pod
spec:
  containers:
  - name: main-container
    image: main-image:latest
    ports:
    - containerPort: 80
  - name: adapter-container
    image: adapter-image:latest
    ports:
    - containerPort: 9090
```

DEPLOYMENTS AND REPLICASETS

Deployments and ReplicaSets are fundamental components in Kubernetes that play an important role in managing the deployment, scaling, and update lifecycle of applications within a cluster.

A Deployment is a higher-level abstraction in Kubernetes that facilitates the declarative definition of application lifecycles.

A ReplicaSet is closely tied to Deployments and acts as a controller, ensuring a specified number of identical pod replicas are running at all times.

VISUAL GUIDE: LIFECYCLE OF DEPLOYMENTS AND HOW REPLICASETS ENSURE HIGH AVAILABILITY

Deployments in Kubernetes manage the deployment and scaling of pods, providing a declarative approach to defining the desired state. The life cycle involves several key stages:

- ¥ **Creation:** A Deployment is created with the desired number of replicas and pod template specifications.
- ¥ **ReplicaSet Generation:** The Deployment creates an associated ReplicaSet, which ensures the specified number of pod replicas are running.
- ¥ **Pod Creation:** The ReplicaSet creates individual pods based on the defined template.
- ¥ **Scaling:** Scaling can occur by adjusting the desired number of replicas. The Deployment ensures the correct number of pods are maintained.
- ¥ **Rolling Updates:** When updating the application, the Deployment creates a new ReplicaSet with the updated configuration while gradually scaling down the old ReplicaSet.
- ¥ **Termination:** The old ReplicaSet is eventually scaled to zero, and the update is complete.

CHEAT SHEET FOR SCALING APPLICATIONS WITH DEPLOYMENTS AND MANAGING ROLLING UPDATES

Scaling Deployments

Scale to a Specific Number of Replicas: `kubectl scale deployment <deployment_name> --replicas=<num>`

Autoscale Based on CPU Utilization: `kubectl autoscale deployment <deployment_name> --cpu-percent=<percentage> --min=<min_replicas> --max=<max_replicas>`

Managing Rolling Updates

Update Deployment with a New Image: `kubectl set image deployment/<deployment_name> <container_name>=<new_image>`

Monitor Rolling Update Progress: `kubectl rollout status deployment/<deployment_name>`

Rollback to Previous Version: `kubectl rollout undo deployment/<deployment_name>`

Pause and Resume Rolling Updates: `kubectl rollout pause deployment/<deployment_name>` and `kubectl rollout resume deployment/<deployment_name>`

Customize Update Strategy (e.g., Canary Deployment): `kubectl apply -f canary-deployment-strategy.yaml`

These commands offer a quick reference for scaling deployments to meet demand and efficiently managing rolling updates. Whether adjusting the number of replicas or orchestrating updates with different strategies, this cheat sheet helps streamline the deployment process within a Kubernetes environment.

CODE EXAMPLES FOR CREATING AND UPDATING REPLICASETS

Creating a ReplicaSet

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: example-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: app-container
          image: example-image:latest
```

¥ This YAML configuration defines a ReplicaSet named example-replicaset.

¥ It specifies that three replicas of the pod should be maintained.

¥ The pod template includes a container named app-container using the example-image:latest image.

Updating a ReplicaSet

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: example-replicaset
spec:
  replicas: 5 # Update the number
  of replicas
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: app-container
          image: updated-example-
            image:latest # Update the container
            image
```

¥ This YAML configuration updates the existing ReplicaSet named example-replicaset.

¥ It modifies the desired number of replicas to 5.

¥ The pod template is updated to use the updated-example-image:latest container image.

SERVICES AND NETWORKING

Services define a stable endpoint for accessing a set of pods, while Networking encompasses the mechanisms that enable effective communication between these pods.

GRAPHICAL REPRESENTATION: KUBERNETES NETWORKING COMPONENTS AND THEIR INTERACTIONS

Figure 2. Kubernetes Networking Components

CHEAT SHEET FOR CREATING AND MANAGING KUBERNETES SERVICES

Creating Services

Create a Service from a YAML file: `kubectl apply -f service-definition.yaml`

Example* service-definition.yaml:*

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

Managing Services

List all Services in the default namespace: `kubectl get services`

Delete a Service: `kubectl delete service <service_name>`

Expose a Deployment as a Service: `kubectl expose deployment <deployment_name> --port=<external_port>`

Service Discovery

Discover service details: `kubectl describe service`

<service_name>

Retrieve the ClusterIP of a Service: `kubectl get service <service_name> -o jsonpath='{.spec.clusterIP}'`

YAML EXAMPLES FOR DEFINING SERVICES AND EXPOSING APPLICATIONS

ClusterIP Service

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

¥ This YAML configuration defines a ClusterIP service named clusterip-service.

¥ It selects pods with the label app: example.

¥ The service exposes port 80 internally and routes traffic to the pods' port 8080.

NodePort Service

```
apiVersion: v1
kind: Service
metadata:
  name: nodeport-service
spec:
  selector:
    app: example
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: NodePort
```

¥ This YAML configuration creates a NodePort service named nodeport-service.

¥ It selects pods labeled app: example.

¥ The service exposes port 80 on each node, forwarding traffic to the selected pods on port 8080.

LoadBalancer Service

```
apiVersion: v1
kind: Service
metadata:
  name: loadbalancer-service
spec:
  selector:
    app: example
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

¥ This YAML configuration defines a LoadBalancer service named loadbalancer-service.

¥ It selects pods labeled app: example.

¥ The service exposes port 80, and in cloud environments, a cloud-specific load balancer is provisioned to distribute traffic to the selected pods.

CONFIGURATION AND SECRETS

ConfigMaps allow central management of non-sensitive configuration details, while Secrets provide a secure mechanism for storing sensitive information such as passwords.

KEY CONFIGURATIONS IN KUBERNETES AND PRACTICAL USE CASES

Configuration	Use Case
ConfigMaps	Centralized configuration management for applications.
Secrets	Securely store sensitive information like passwords.

Configuration	Use Case
Resource Limits	Restrict resource consumption by containers.
Environment Variables	Pass configuration details to containers.
Volume Mounts	Share files and directories between containers and pods.
Labels and Annotations	Metadata for organizing and querying objects in Kubernetes.

SECURE PRACTICES FOR MANAGING AND CONSUMING SECRETS

- Use Kubernetes Secrets: Leverage the built-in Kubernetes Secrets API to store sensitive information securely.
- Encrypt Data in Transit: Ensure that data exchanged between components is encrypted, especially when dealing with sensitive information.
- Role-Based Access Control (RBAC): Implement RBAC to control access to Secrets, allowing only authorized entities to retrieve sensitive data.
- Avoid Hardcoding Secrets: Refrain from hardcoding secrets directly in application code or configuration files.
- Regularly Rotate Secrets: Periodically rotate passwords and cryptographic keys to minimize the impact of a potential security breach.

CODE SNIPPETS FOR USING CONFIGMAPS AND SECRETS IN KUBERNETES PODS

Using ConfigMaps in a Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod
spec:
  containers:
  - name: app-container
    image: app-image:latest
    envFrom:
```

```
  - configMapRef:
    name: example-configmap
```

- This YAML configuration defines a pod named configmap-pod.
- The envFrom field references a ConfigMap named example-configmap.
- The values from the ConfigMap are injected as environment variables into the pod.

Using Secrets in a Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-pod
spec:
  containers:
  - name: app-container
    image: app-image:latest
    envFrom:
    - secretRef:
      name: example-secret
```

- This YAML configuration creates a pod named secret-pod.
- The envFrom field references a Secret named example-secret.
- The values from the Secret are injected as environment variables into the pod.

Creating ConfigMap

```
kubectl create configmap example-configmap --from-literal=key1=value1 --from-literal=key2=value2
```

- This command creates a ConfigMap named example-configmap.
- It includes key-value pairs (key1=value1 and key2=value2) that can be accessed by pods referencing this ConfigMap.

MONITORING AND TROUBLESHOOTING

Monitoring and troubleshooting in Kubernetes involve the continuous observation and resolution of issues within a cluster. Monitoring tools like

Prometheus, Grafana, and cAdvisor provide insights into cluster health, while troubleshooting involves identifying and addressing common issues in pods, nodes, networking, and services.

CHEAT SHEET FOR TROUBLESHOOTING COMMON ISSUES IN KUBERNETES CLUSTERS

¥ Pod Issues:

- ! Check pod logs: `kubectl logs <pod_name>`
- ! Describe pod for detailed info: `kubectl describe pod <pod_name>`

¥ Node Issues:

- ! View nodes: `kubectl get nodes`
- ! Check node events: `kubectl describe node <node_name>`

¥ Networking Issues:

- ! Inspect network policies: `kubectl get networkpolicies`

¥ Service Issues:

- ! Verify service status: `kubectl get services`

¥ Configuration Issues:

- ! Check ConfigMaps: `kubectl get configmaps`

¥ Resource Issues:

- ! View resource usage: `kubectl top nodes` or `kubectl top pods`

¥ Cluster Information:

- ! Get cluster information: `kubectl cluster-info`

¥ API Resources:

- ! List available API resources: `kubectl api-resources`

¥ Version Information:

- ! Check Kubernetes version: `kubectl version`

VISUAL REPRESENTATION OF KUBERNETES DASHBOARD COMPONENTS

Figure 3. Kubernetes Dashboard

ADVANCED TOPICS

CHEAT SHEET FOR HELM CHARTS AND EFFICIENT PACKAGE MANAGEMENT IN KUBERNETES

Install Helm: `curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash`

Add Helm Repository: `helm repo add stable https://charts.helm.sh/stable`

Install a Helm Chart: `helm install my-release stable/<chart_name>`

Upgrade a Helm Release: `helm upgrade my-release stable/<chart_name>`

Uninstall a Helm Release: `helm uninstall my-release`

CODE EXAMPLES FOR SETTING UP PERSISTENT VOLUMES AND PERSISTENT VOLUME CLAIMS

Define Persistent Volume (example-pv.yaml)

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /path/to/host/folder
```

Defines a Persistent Volume named example-pv with a capacity of 1Gi using a host path.

Create Persistent Volume Claim (example-pvc.yaml)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Creates a Persistent Volume Claim named example-pvc requesting 1Gi storage with ReadWriteOnce access.

Use Persistent Volume Claim in Pod (example-pod.yaml)

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: app-container
      image: app-image:latest
      volumeMounts:
        - mountPath: /app/data
          name: data-volume
  volumes:
    - name: data-volume
      persistentVolumeClaim:
        claimName: example-pvc
```

¥ Defines a Pod named example-pod with a container using the specified image.

¥ Mounts a volume (data-volume) using the Persistent Volume Claim (example-pvc).

ADDITIONAL COMMANDS

FOR NODE MANAGEMENT

Nodes in Kubernetes are the physical or virtual machines that form the cluster's computing infrastructure. These commands allow users to inspect, label, and manage nodes for optimal cluster performance.

Command	Description
<code>kubectl get nodes</code>	Display information about nodes in the cluster.
<code>kubectl describe node <node_name></code>	Show detailed information about a specific node.
<code>kubectl label nodes <node_name> <label></code>	Attach a label to a node for categorization.
<code>kubectl cordon <node_name></code>	Mark a node as unschedulable for maintenance.
<code>kubectl uncordon <node_name></code>	Allow a previously unschedulable node to schedule pods.
<code>kubectl drain <node_name></code>	Safely evict all pods from a node before maintenance.

FOR NAMESPACE MANAGEMENT

Namespaces in Kubernetes are virtual clusters within a physical cluster, providing isolation and organization. These commands facilitate the management of namespaces, enabling users to create and inspect them.

Command	Description
<code>kubectl get namespaces</code>	List all namespaces in the cluster.
<code>kubectl create namespace <namespace_name></code>	Create a new namespace.
<code>kubectl describe namespace <namespace_name></code>	Show details about a specific namespace.

SERVICE COMMANDS

Services in Kubernetes provide a stable endpoint to access a set of pods. These commands aid in the management of services, allowing users to expose deployments and handle service configurations.

Command	Description
<code>kubectl get services</code>	List all services in the default namespace.
<code>kubectl get services --all-namespaces</code>	List all services in all namespaces.
<code>kubectl describe service <service_name></code>	Display detailed information about a service.
<code>kubectl expose deployment <deployment_name> --port=<external_port></code>	Expose a deployment as a service.
<code>kubectl delete service <service_name></code>	Delete a specific service.

DEPLOYMENT COMMANDS

Deployments in Kubernetes define the desired state for a set of pods. These commands assist in the management of deployments, allowing users to scale, inspect, and modify deployment configurations.

Command	Description
<code>kubectl get deployments</code>	List all deployments in the default namespace.
<code>kubectl get deployments --all-namespaces</code>	List all deployments in all namespaces.
<code>kubectl describe deployment <deployment_name></code>	Show details about a specific deployment.
<code>kubectl scale deployment <deployment_name> --replicas=<num></code>	Scale the number of replicas in a deployment.

In this comprehensive Kubernetes cheat sheet, we've delved into fundamental concepts, advanced techniques, and essential commands for efficient cluster management. From basic terminology and architecture to intricate configurations, monitoring, and troubleshooting, this resource serves as a valuable quick reference.

As your next steps, consider applying these learnings in practical scenarios, experimenting with different configurations, and exploring additional Kubernetes features. Stay informed about updates in the Kubernetes ecosystem, continuously refine your skills, and consider contributing to the community.

RESOURCES

These resources cover a wide range of topics and cater to different levels of expertise, helping you deepen your understanding of Kubernetes.

¥ Kubernetes Monitoring

- ! Website: Understanding [Kubernetes Monitoring](#)
- ! A comprehensive guide that covers the basics and advanced concepts of Kubernetes Monitoring

¥ Kubernetes monitoring tools

- ! Website: Best [Kubernetes monitoring tools](#)
- ! A Listicle article that listed 13+ opensource and free Kubernetes monitoring tools

¥ Kubernetes Logging

- ! Website: [Kubernetes Logging](#) 101: A Complete Guide
- ! The article talks about K8s logging, various logging methods, best practices and tools in an easy to read way.

¥ Official Kubernetes Documentation

- ! Website: [Kubernetes Documentation](#)
- ! The official site of Kubernetes.

¥ Official Kubernetes Blog

- ! Website: [Kubernetes Blog](#)
- ! Learn all about Kubernetes, be it logging, monitoring, configuring, or anything else.

¥ Kubectl Cheat Sheet

- ! Website: [kubectl Quick Reference](#)
- ! A handy reference for kubectl commands, the primary CLI tool for interacting with Kubernetes clusters.

¥ Kubernetes Patterns

- ! Website: [Kubernetes Patterns, 2nd Edition](#)

! This eBook, offers design patterns and best practices for building applications on Kubernetes.

¥ Kubernetes Cheat Sheet by Red Hat

! Website: [Red Hat's Kubernetes Cheat Sheet](#)

! This concise cheat sheet covers essential Kubernetes commands and concepts.

¥ Kubernetes Podcast

! Website: [Kubernetes Podcast by Google](#)

! Offers insights, news, and discussions on Kubernetes and its ecosystem.

¥ Kubernetes Slack Channels and Forums

! Slack: [Kubernetes Slack Channels](#)

! Reddit: [Kubernetes Reddit](#)

! Join Kubernetes-related Slack channels and forums to engage with the community, ask questions, and stay updated on the latest developments.

JCG delivers over 1 million pages each month to more than 700K software developers, architects and decision makers. JCG offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

Copyright © 2014 Exelixis Media P.C. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

CHEATSHEET FEEDBACK
WELCOME
support@javacodegeeks.com

SPONSORSHIP
OPPORTUNITIES
sales@javacodegeeks.com