Use Case: PokéDelivery – Serverless Food für Pokémon

Du bist Teil des PokéDelivery-Teams, einem innovativen Start-up, das Pokémon in ganz Kanto mit frischem Sushi versorgt. Die Plattform basiert auf modernen Prinzipien wie Automatisierung, Skalierbarkeit und Ausfallsicherheit.

Um unsere Lieferungen noch besser auf die Vorlieben unserer Pokémon-Kundschaft abzustimmen (Pikachu liebt Lachs!), entwickeln wir eine Data Retrieval Platform. Diese Plattform soll strukturierte Informationen über Pokémon abrufen und bereitstellen. Sie bildet die Grundlage für datenbasierte Entscheidungen und personalisierte Services.

Deine Aufgabe ist es, eine Anwendung zu entwickeln, die über eine standardisierte Schnittstelle Daten zu einzelnen Pokémon liefert. Die Plattform soll lokal und in der Cloud betrieben werden können, kontinuierlich getestet und automatisiert bereitgestellt werden.

Systemanforderungen – PokéDelivery

Das Dev Team hat bereits eine Entwicklerversion der APP auf Basis des `Requirements 2` für euch zur Verfügung gestellt

1. **Systemumgebung**

- Das System muss auf einem ARM-Gerät/VM (RaspberryPi oder B2PTS v2 VM) mit Ubuntu 22.04 LTS betrieben werden können.
- Alle eingesetzten Softwarekomponenten müssen ARM64-kompatibel sein.

---

2. **Applikation**

- Die Anwendung muss eine REST-API bereitstellen, die Informationen zu Pokémon liefert.
- Die API muss mindestens folgende Endpunkte unterstützen:
  - `GET /pokemon/{name}`
- Die Anwendung muss als Azure Function lokal und in der Cloud lauffähig sein.

---

3. **Codeverwaltung**

- Der Quellcode muss in einem zentralen Repository verwaltet werden.
- Es muss eine Branching-Strategie definiert und dokumentiert sein.
- Änderungen am Code dürfen nur nach erfolgreichem Review und bestandener CI integriert werden.

---

4. **Continuous Integration**

- Jede Änderung im Repository muss automatisch einen Build- und Testprozess auslösen.
- Die CI-Pipeline muss automatisierte API-Tests ausführen.
  - Die API-Tests sind mit einem geeigneten Tool/Bibliothek zu entwerfen und zu implementieren.
- Die CI-Umgebung muss auf ARM64-Architektur lauffähig sein.

---

5. **Security & Compliance**

- Es dürfen keine Zugangsdaten, Tokens oder Secrets im Quellcode-Repository gespeichert werden.
- Die eingesetzten Bibliotheken müssen regelmäßig auf Sicherheitslücken geprüft und aktualisiert werden.
- Der Zugriff auf Continuous Integration Tool darf beliebig gestaltet werden (falls nicht identisch mit dem CD-System)
- Der Zugriff auf Continuous Deployment-Systeme muss authentifiziert und rollenbasiert erfolgen.
    - "anonyme" User dürfen gar keinen Zugriff haben.
    - "manager" User dürfen nur lesenden Zugriff haben.
    - "developer" User sollen vollen Zugriff aufs System haben.
- Continuous Deployment soll mittels Jenkins erfolgen, da der Kunde dort den größten Wissensstand hat.

---

6. **Containerisierung & Jenkins**

- Jenkins muss als Container auf ARM64 lauffähig sein.
- Die Jenkins-Konfiguration muss möglichst vollständig als Code vorliegen.
- Es müssen mindestens zwei Jenkins-Agents auf unterschiedlichen Geräten/VMs eingerichtet werden.
- Jenkins sollte keine Warnungen mehr anzeigen (Security).
- Die Agent-Konfiguration muss automatisiert erfolgen (bevorzugt via Ansible).

---

7. **Monitoring & Observability**

- Jenkins muss interne Metriken bereitstellen.
- Diese Metriken erfasst/gesammelt werden.
- Ein Dashboard zur Visualisierung der CI/CD-Metriken muss bereitgestellt werden.
- Logs aller relevanten Container (Jenkins, Mtriken, Dashboards) müssen zentral einsehbar sein.

---

8. **Deployment**

- Die Anwendung muss über Jenkins manuell in Azure deploybar sein.
- Das Deployment erfolgt mit "Infrastructure as Code" (IaC).
- Die Anwendung muss Metriken bereitstellen, die über Azure Application Insights analysierbar sind.

9. **Dokumentation**

- Eine Dokumentation der Architektur soll vorliegen.
- Entscheidungen für oder gegen bestimmte Architektur-Entscheidungen und Tools soll vorliegen.
- Eine Entwickler Dokumentation soll vorliegen.

Wissenswertes

Videos

🎥 **Kursname**	🔗 **URL**	🎯 **Fokus im Bootcamp**

Kursname	URL	Fokus im Bootcamp
GitHub Actions	https://capgemini.udemy.com/course/github-actions-the-complete-guide/	Git-Crashkurs, Basics, Events
AZ-900: Microsoft Azure Fundamentals	https://capgemini.udemy.com/course/az900-azure/	gern komplett, Fokus auf Allgemeines Verständnis, Compute & Storage
Jenkins	https://capgemini.udemy.com/course/jenkins-masterclass/	Komplett durchgehen
Ansible	https://capgemini.udemy.com/course/learn-ansible/	Komplett durchgehen
Terraform	https://capgemini.udemy.com/course/terraform-for-the-absolute-beginners/	Komplett durchgehen
Docker	https://capgemini.udemy.com/course/learn-docker/	komplett
Cloud Native, Containers, Kubernetes	https://capgemini.udemy.com/course/dive-into-cloud-native-containers-kubernetes-and-the-kcna/	Ohne Docker; Kubernetes empfohlen

Weitere Informationen:

Tool	Offizielle Dokumentation / Info
GitHub Actions	https://docs.github.com/en/actions
Dependabot	https://docs.github.com/en/code-security/supply-chain-security/keeping-your-dependencies-updated-automatically
Azure Functions	https://learn.microsoft.com/en-us/azure/azure-functions/
Azure Functions Core Tools	https://learn.microsoft.com/en-us/azure/azure-functions/functions-run-local
Bruno (API Testing)	https://github.com/usebruno/bruno
Docker Compose	https://docs.docker.com/compose/
Jenkins	https://www.jenkins.io/doc/
Ansible	https://docs.ansible.com/
Terraform	https://developer.hashicorp.com/terraform/docs
Portainer	https://docs.portainer.io/
Prometheus	https://prometheus.io/docs/introduction/overview/
Grafana	https://grafana.com/docs/

🛠 Tool	🔗 Offizielle Dokumentation / Info
Azure Portal	https://portal.azure.com/#home
Application Insights	https://learn.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview