**Summary**

In this 15-minute paired activity, teams will design a portable web app by selecting cloud-agnostic service architectures, choosing flexible data storage options, and implementing multi-cloud deployment strategies. Cloud-agnostic development maximizes portability by avoiding vendor-specific dependencies and leveraging standard open-source technologies. Data storage should balance ACID-compliant relational databases with horizontally scalable NoSQL stores or Backend-as-a-Service platforms to meet diverse application needs. The deployment strategy emphasizes containerization with Docker, orchestration via Kubernetes for true multi-cloud flexibility, and Infrastructure as Code using Terraform to automate resource provisioning across AWS, Azure, and beyond.

**Service Choices**

**Cloud-Agnostic Frameworks**

Cloud-agnostic architectures are intentionally designed to run on any cloud without major code changes. Teams should pick standard, widely supported frameworks—such as React for the frontend and Node.js with Express or Python with FastAPI for the backend—to ensure code portability and a large ecosystem of plugins and tooling.

**Microservices & BaaS**

Adopting a microservice architecture decouples functionality into independently deployable services, simplifying cloud migrations and scaling. For rapid prototyping or when you want to offload backend maintenance, consider Backend-as-a-Service offerings like Firebase or AWS Amplify, which provide user authentication, real-time databases, and storage via unified SDKs—boosting portability and reducing custom infrastructure work ([Backend as a service](https://en.wikipedia.org/wiki/Backend_as_a_service?utm_source=chatgpt.com)).

**Data Storage Options**

**Relational vs. NoSQL**

Relational databases (e.g., PostgreSQL, MySQL) offer structured schemas and strong ACID guarantees, making them ideal for transactional workloads; they can be deployed via managed services or containerized clusters across clouds. NoSQL databases (e.g., MongoDB Atlas, Azure Cosmos DB) provide schema flexibility and native horizontal scalability, enabling easy global distribution and high-availability configurations in multi-cloud environments

**Backend-as-a-Service Platforms**

For teams that need to accelerate development, BaaS platforms abstract common backend services—database management, file storage, authentication, and notifications—behind a consistent API. Firebase and AWS Amplify both support web and mobile clients, ensuring that the same service endpoints work regardless of deployment target.

**Deployment Strategy**

**Containerization & Orchestration**

Package your frontend and backend services into Docker containers to guarantee environment consistency across development, testing, and production. Use Kubernetes to orchestrate these containers, enabling seamless deployments on any managed Kubernetes service (AKS, EKS, GKE) and achieving true multi-cloud portability.

**Infrastructure as Code & CI/CD**

Leverage Terraform to define and provision infrastructure across AWS, Azure, and other providers with a single declarative workflow; use modules and remote state backends to maintain reusable, version-controlled configurations ([Multi-cloud provisioning - Terraform](https://www.terraform.io/use-cases/multi-cloud-deployment?utm_source=chatgpt.com)). Integrate your Terraform runs into a CI/CD pipeline (e.g., using Harness, GitHub Actions, or Jenkins) to automate plan, apply, and destroy stages—ensuring reliable, repeatable multi-cloud deployments ([Multi-Cloud Continuous Delivery - Harness](https://www.harness.io/harness-devops-academy/multi-cloud-continuous-delivery?utm_source=chatgpt.com)).