**Todo Application with Modern Internal Developer Platform Demo**

**Introduction (1 minute)**

"Hello everyone. Today I'll be demonstrating a complete DevOps project that combines a Todo application with a Modern Internal Developer Platform. This project fulfills both the Observability and Cloud Deployment assignment requirements, as well as implementing Track B's Developer Platform specifications."

"My implementation integrates state-of-the-art observability tooling with cloud-native deployment strategies and a comprehensive developer platform to showcase modern DevOps practices."

**Part 1: Application Overview (2 minutes)**

**1.1 Application Functionality**

"Let me start by showing you the core application - a full-stack Todo list built with React and Node.js."

**Demo Actions:**

* Navigate to the Todo application frontend
* Create a new todo item: "Complete DevOps assignment"
* Edit an item to add more details
* Mark an item as complete
* Delete an item

"The application allows users to create, edit, complete, and delete todo items. It's intentionally simple in functionality, but it serves as an excellent foundation for demonstrating DevOps principles."

**1.2 Application Architecture**

"The application follows a typical three-tier architecture:"

**Demo Actions:**

* Show the architecture diagram
* Highlight the application components

"The frontend is built with React and communicates with our Express.js backend API. The backend stores data in MongoDB. What makes this implementation special is how we've instrumented it for observability and prepared it for cloud deployment."

**Part 2: Assignment Requirements - Observability (3 minutes)**

**2.1 OpenTelemetry Instrumentation**

"For the observability component of the assignment, I've implemented OpenTelemetry throughout the stack."

**Demo Actions:**

* Show the tracing.js files in both frontend and backend
* Highlight the key instrumentation code

"OpenTelemetry provides a vendor-neutral way to collect traces, metrics, and logs. I've added instrumentation to both the frontend and backend, capturing everything from page loads to database operations."

**2.2 Metrics with Prometheus and Grafana**

"Let's look at the metrics we're collecting using Prometheus and visualizing with Grafana."

**Demo Actions:**

* Navigate to Grafana dashboard
* Show the main application dashboard with key metrics:
  + HTTP request rate
  + Response times (p95, p99)
  + Error rates (4xx, 5xx)
  + Node.js performance metrics

"Here we can see our application metrics. The assignment required us to track HTTP request rates, error rates, and latency percentiles, which you can see here. I've also added custom application metrics to track specific business operations."

**2.3 Distributed Tracing with Jaeger**

"For distributed tracing, I've implemented Jaeger to visualize request flows."

**Demo Actions:**

* Navigate to Jaeger UI
* Search for a trace
* Show the span details for a complete request

"Jaeger allows us to trace requests across service boundaries. Here's a trace showing a todo creation operation, from frontend to backend to database and back. This makes it much easier to troubleshoot performance issues and understand system behavior."

**Part 3: Assignment Requirements - Cloud Deployment (3 minutes)**

**3.1 Containerization**

"Our application is fully containerized using Docker."

**Demo Actions:**

* Show the Dockerfiles for frontend and backend
* Highlight multi-stage builds and optimizations

"I've created optimized Dockerfiles for both frontend and backend. These containers are designed to be lightweight, secure, and production-ready."

**3.2 Infrastructure as Code with Terraform**

"For cloud infrastructure, I've prepared Terraform scripts to provision AWS resources."

**Demo Actions:**

* Show the Terraform configuration files
* Highlight the main resources:
  + VPC setup
  + EKS cluster configuration
  + IAM roles

"These Terraform scripts will provision an EKS cluster with all the necessary networking, security, and identity components. The infrastructure is defined as code, making it repeatable and version-controlled."

**3.3 Kubernetes Deployment**

"Once the infrastructure is provisioned, our application is deployed using Kubernetes."

**Demo Actions:**

* Show the Kubernetes manifests
* Highlight key features:
  + Multi-environment support (dev/prod)
  + Readiness/liveness probes
  + Resource limits
  + ConfigMaps and Secrets

"These Kubernetes manifests define how our application runs in the cluster. I've included configurations for multiple environments, health checks, and proper resource management."

**Part 4: Project Track B - Internal Developer Platform (5 minutes)**

**4.1 Developer Portal with Backstage**

"Now, let's explore the Internal Developer Platform components, starting with Backstage, our developer portal."

**Demo Actions:**

* Show the Backstage catalog files
* Explain the service catalog structure
* Highlight API documentation

"Backstage provides a centralized interface for developers to discover services, APIs, and documentation. I've created catalog entries for our frontend, backend, and database components, making them easily discoverable."

**4.2 GitOps with ArgoCD**

"For GitOps-based continuous deployment, I've implemented ArgoCD."

**Demo Actions:**

* Show the ArgoCD application configuration
* Explain the GitOps workflow
* Demonstrate the deployment process

"ArgoCD ensures that our deployed applications always match what's defined in our Git repository. When we push changes to the repository, ArgoCD automatically synchronizes our cluster state to match."

**4.3 Policy Enforcement with OPA/Gatekeeper**

"To ensure governance and compliance, I've added OPA/Gatekeeper."

**Demo Actions:**

* Show the policy definitions
* Highlight resource requirements policy
* Demonstrate policy validation

"These policies enforce organizational standards like requiring resource limits and specific labels. This prevents deployment of resources that don't meet our quality and security standards."

**4.4 Platform Integration**

"The real power comes from how these components work together to form a cohesive platform."

**Demo Actions:**

* Show the updated architecture diagram with platform components
* Explain the integration points
* Highlight the developer experience

"A developer using this platform can easily create new services from templates in Backstage, deploy them using CI/CD pipelines, monitor them with our observability tools, and ensure they meet organizational standards through policy enforcement."

**Part 5: Demo Scenario - End-to-End Developer Experience (3 minutes)**

"Let me demonstrate how a developer would use this platform by walking through a typical workflow."

**5.1 Creating and Deploying a Change**

**Demo Actions:**

* Make a simple code change to the Todo application
* Commit the change to GitHub
* Show the CI pipeline running
* Demonstrate ArgoCD detecting and applying the change

"When a developer commits a change, our CI pipeline automatically runs tests and builds a new container image. Then ArgoCD detects the new version and deploys it to our cluster."

**5.2 Monitoring the Deployment**

**Demo Actions:**

* Show the deployment process in real-time
* Monitor metrics during deployment
* Inspect traces for the new version

"During deployment, we can monitor our application health in real-time using our observability tools. We can see that traffic is gradually shifting to the new version, and we can verify that performance remains within acceptable bounds."

**5.3 Policy Validation**

**Demo Actions:**

* Attempt to deploy a resource that violates policies
* Show the rejection and error message
* Fix the violation and deploy successfully

"If we try to deploy something that violates our policies, like a container without resource limits, the platform prevents the deployment and provides clear feedback. After fixing the issue, the deployment proceeds successfully."

**Conclusion (1 minute)**

"In this demo, I've shown how I've met both the assignment requirements for observability and cloud deployment, as well as implemented a comprehensive Internal Developer Platform according to Track B specifications."

"The platform provides a complete environment for developing, deploying, monitoring, and managing applications following modern DevOps practices. It includes:"

1. Comprehensive observability with OpenTelemetry, Prometheus, Grafana, and Jaeger
2. Cloud-native deployment with Docker, Kubernetes, and Terraform
3. A developer platform with Backstage, ArgoCD, and OPA/Gatekeeper

"This implementation demonstrates not just the technical components, but how they work together to create a cohesive, efficient developer experience."

"Thank you for your attention. I'm happy to answer any questions about the implementation, design decisions, or potential enhancements."

**Q&A Preparation**

**Potential Questions and Answers:**

**Q: How would you scale this platform for a larger organization?**  
A: For larger organizations, I would add more components like service mesh for advanced networking, secrets management with Vault, and expand Backstage with more plugins. I would also implement multi-cluster support and potentially multi-cloud capabilities.

**Q: Why did you choose MongoDB over a relational database?**  
A: MongoDB's document model aligns well with the JSON-based nature of our application data. For a Todo app, we don't need complex relationships or transactions, so MongoDB's flexibility and scalability benefits outweigh the advantages of a relational database in this case.

**Q: How would you implement Blue/Green deployments in this platform?**  
A: I would use Kubernetes and Istio features to implement Blue/Green deployments. With Istio's traffic management capabilities, we could route traffic between different versions of our application and gradually shift from the old (blue) to the new (green) version once it's validated.

**Q: What was the most challenging part of implementing observability?**  
A: Ensuring consistent context propagation across service boundaries was challenging, particularly between frontend and backend. We needed to carefully configure our OpenTelemetry propagators and ensure that trace IDs were properly passed in HTTP headers.

**Q: How would you handle database migrations in a production environment?**  
A: For MongoDB schema evolution, I would implement backward-compatible changes in the application code. For more structured migrations, I would integrate a tool like MongoDB Migrations to manage schema changes safely with version control and rollback capabilities.

**Q: How does your platform handle secrets management?**  
A: Currently, we use Kubernetes Secrets which are appropriate for the demonstration. In a production environment, I would integrate HashiCorp Vault or AWS Secrets Manager, and add a secrets injection mechanism into our deployment process.

**Q: What additional security measures would you implement for a production deployment?**  
A: I would add network policies to restrict pod-to-pod communication, implement pod security policies, use image scanning in the CI pipeline, implement mTLS with Istio, and set up regular security scanning of the cluster with tools like Trivy or Falco.