Slide Deck: Lecture 8 - Deployment View & Quality Attribute Analysis (ATAM)

1. Title Slide: Finalizing the Blueprint

- **Title:** Software Architecture: Lecture 8
- **Subtitle:** Deployment View & Quality Attribute Analysis (ATAM)
- **Focus:** Documenting the final structure and evaluating how well the architecture meets the **ASRs** defined in Lecture 1.

---

2. Recap: The Final Architecture

- **Pattern: Microservices Architecture** utilizing **EDA**.
- **Components:** API Gateway (5000), Product Service (5001), Order Service, Notification Service, and the RabbitMQ Broker.
- **Goal Today:** Move from the *logical* view (components) to the *physical* view (servers) and critically analyze the trade-offs.

---

3. Knowledge Base: The Physical View (Deployment Diagram)

- **Purpose:** Shows the physical mapping of software artifacts (components, services) onto execution environments (**Nodes**).
- **Artifact:** The **UML Deployment Diagram**.
- **Key Elements:**
    1. **Node (3D Box):** A hardware or software execution environment (Server, Cluster, Docker Container).
    2. **Artifact (Document Icon):** A file or executable component deployed to a node (e.g., product_service.py).
    3. **Communication Path (Line):** Represents network connections (e.g., HTTP/S, AMQP).

---

4. Activity 1: Defining the Physical Nodes (ShopSphere)

- **Goal:** Define the hardware/software environments for our services.

Node Name	Node Type	Purpose	Hosted Artifacts
Edge Network	Server	External facing, load balancing.	API Gateway
Application Cluster	Cluster/VMs	Scalable environment for core logic.	Product Service, Order Service
Messaging Server	Server	Dedicated asynchronous routing.	RabbitMQ Broker
Database Server	Server	Permanent state storage.	SQL/NoSQL Databases

---

5. Practical Activity (Hands-On): Drawing Nodes

- **Tool:** draw.io (UML Deployment Stencil).
- **Steps:**
    1. Draw the three main 3D boxes: **Edge Network**, **Application Cluster**, and **Messaging Server**.
    2. Use the <<device>> stereotype for physical hardware or <<execution environment>> for software environments (e.g., Docker container).

---

6. Activity 2: Mapping Artifacts to Nodes

- **Goal:** Place the executable services developed in previous lectures onto the nodes.
- **Artifact Mapping:**
    - **Edge Network** hosts the **API Gateway** (gateway.py).
    - **Application Cluster** hosts the **Product Service** (product_service.py) and **Order Service** (order_service.py).
    - **Messaging Server** hosts the **RabbitMQ Broker** artifact.

---

7. Activity 3: Modeling Communication Paths

- **Goal:** Show how the nodes talk to each other.
- **Paths:**

1. **Client $\leftrightarrow$ Edge Network:** HTTP/S (Internet Traffic).
2. **Edge Network $\rightarrow$ Application Cluster:** HTTP (Internal Routing).
3. **Application Cluster $\leftrightarrow$ Messaging Server:** AMQP (Asynchronous Events).

---

8. Visualization: The UML Deployment Diagram

---

9. Knowledge Base: Quality Attribute Analysis (ATAM)

- **ATAM (Architecture Trade-off Analysis Method):** A systematic approach to evaluate how well an architecture satisfies its quality attributes (ASRs).
- **Core Idea:** You can't maximize all qualities (e.g., maximizing security often reduces performance). ATAM highlights these **trade-offs**.
- **Tool:** Using **Scenarios** to stress-test the architecture mentally.

---

10. The Scenario Structure

- A good scenario describes:
    - **Source:** Who initiates the action (e.g., 10,000 Concurrent Users).
    - **Stimulus:** The event (e.g., A spike in traffic).
    - **Artifact:** The part of the system affected (e.g., Product Catalog).
    - **Environment:** Conditions (e.g., During Black Friday).
    - **Response:** The expected result (e.g., Latency remains below 2 seconds).

---

11. Activity 4: Defining Core Scenarios

- **Goal:** Create two scenarios based on our key ASRs (Scalability and Availability).

Scenario ID	ASR	Scenario Statement
SS1	**Scalability**	**Source:** 10,000 Concurrent Users. **Stimulus:** Search the product catalog. **Response:** Latency remains under 2 seconds.
AS1	**Fault Isolation**	**Source:** Inventory Service fails. **Stimulus:** Customer places an order. **Response:** Order is confirmed, and the failure is handled asynchronously.

---

12. Analysis: Scenario SS1 (Scalability)

- **Architecture Tested:** Microservices (Product Service).
- **Step-by-Step Response:**
    1. Traffic hits the **API Gateway** (Edge Network).
    2. Traffic is routed to the **Product Service** cluster.
    3. Because the Product Service is isolated, the load only impacts the **Application Cluster** nodes running that service.
    4. The system administrator **horizontally scales** only the Product Service instances.
- **Result:** The architecture **succeeds**. It efficiently scales the high-load component without touching the Order or Notification services.

---

13. Analysis: Scenario AS1 (Fault Isolation)

- **Architecture Tested:** EDA (Order Service $\rightarrow$ Message Broker $\rightarrow$ Inventory/Notification).
- **Step-by-Step Response:**
    1. The **Order Service** receives the order.
    2. It attempts synchronous communication (REST) to check user credentials (User Service) – **Success**.
    3. It publishes an OrderPlacedEvent to the **RabbitMQ Broker**.
    4. The **Inventory Service** (Consumer) is down.

5. The Broker holds the message, and the Order Service **returns success** to the client immediately.

- **Result:** The architecture **succeeds**. The core transaction is protected by the Broker, meeting the **Fault Isolation ASR**.

---

14. Critical Trade-Off: Simplicity vs. Resilience

- **The Monolith (Lectures 2-3):** Simple development, low operational overhead, but **zero fault isolation** and **expensive scaling**.
- **Microservices + EDA (Lectures 4-7):** High operational complexity (managing 5+ services, queues, and gateways), but **excellent scalability** and **high resilience**.
- **Conclusion:** We made a necessary trade-off: **We sacrificed simplicity to achieve the ASRs**.

---

15. Architectural Conclusion: Final Decisions

- **Best Fit Pattern:** Microservices with API Gateway and Asynchronous Messaging (EDA).
- **Rationale:** This pattern is the only one that satisfies all three critical ASRs: Scalability (via independent services), Fault Isolation (via EDA), and Security (via API Gateway).

---

16. Project Integration: From Code to Documentation

- **Reminder:** Architecture is primarily about the **decisions** and **documentation**, not just the code.
- The code in Lectures 3, 5, 6, and 7 served as the **Proof of Concept** that the designed architecture *can* be built and *will* function as intended under stress (e.g., 503 handling, asynchronous processing).

---

17. Self-Correction: Deployment Diagram Errors

- **Error 1:** Drawing the communication line between two Artifacts without showing the containing Nodes. **(Incorrect)** The line must connect the Nodes, or show the Artifacts connected *inside* the Node.

- **Error 2:** Confusing the Deployment View (physical) with the Component View (logical). **(Key Difference)** Artifacts are code files; Components are logical modules.

---

18. Final Review: Summary of Architectural Learnings

- **Layering (Lecture 2):** Solves **Maintainability** and **Separation of Concerns**.
- **Microservices (Lecture 4):** Solves **Scalability** and **Independent Deployment**.
- **API Gateway (Lecture 6):** Solves **Centralized Security** and **Client Complexity**.
- **EDA (Lecture 7):** Solves **Fault Isolation** and **Temporal Decoupling**.

---

19. Summary of Deliverables for Submission

- **Document 1:** The final **UML Deployment Diagram**, showing all nodes, artifacts, and communication paths.
- **Document 2:** The completed **ATAM Analysis Table** summarizing the results of Scenarios SS1 and AS1.
- **Document 3:** A final **Reflection Paragraph** stating the main **Trade-off** made (Simplicity vs. Resilience) and why it was necessary.

---

20. Q&A and Course Conclusion

- **Questions?** (Final review of the Monolith vs. Microservices comparison).
- **Course Summary:** You now have the tools to design, document, and implement architectures from simple Monoliths to complex distributed systems.
- **Next Steps:** Final Report Submission (integrating all 8 lecture deliverables).