SYSTEMS · TAUGHT WITH KUBERNETES

Distributed Systems

Partial failure, ordering, replication, and consensus, taught with the papers rather than around them. You break a running cluster on purpose and explain precisely why it behaved that way.

Advanced | 4 phases | 22 sessions | 33 contact hours

Syllabus

4 phases, 22 sessions. Each phase ends in something you have built.

1 Partial Failure and Time

Reason correctly about a call that never returns and a clock that is quietly wrong.

5 sessions
$108 alone

1.1	**The Network Is Not There**	Timeouts, the two generals problem, and the three things a failed RPC can actually mean. FREE PREVIEW
1.2	**Clocks Lie**	NTP drift, leap seconds, and why last-write-wins on wall-clock timestamps silently drops writes.
1.3	**Happens-Before by Hand**	Lamport and vector clocks traced on paper, then used to detect concurrent writes in a real log.
1.4	**Failure Detectors and Timeouts**	Heartbeats, phi accrual, gray failure, and the tradeoff between detecting fast and detecting wrong.
1.5	**FLP, and What It Costs**	Why consensus is impossible in an asynchronous model, and the partial synchrony real systems assume.

By the end: **A written failure model for one two-service call path, plus a vector clock tool that flags concurrent writes in a captured log.**

2 Replication and Consensus

Read the Raft paper closely enough to implement it, and say precisely which consistency model you are buying.

6 sessions
$108 alone

2.1	**Consistency Models, Precisely**	Linearizable, sequential, causal, and eventual, defined by what histories each one forbids.
2.2	**CAP, Priced Out**	CAP as a latency bill rather than a slogan, plus PACELC and the cost when there is no partition.
2.3	**Raft, Read Closely**	Terms, elections, and log matching, working through the safety argument line by line.
2.4	**Implement the Log**	AppendEntries, commit index, and the election restriction that stops a stale leader overwriting.
2.5	**Snapshots and Membership**	Log compaction, learners, joint consensus, and the reconfiguration bug most implementations ship.
2.6	**What etcd Actually Gives You**	etcd 3.7 revisions, watches, and leases, and how kube-apiserver decides when to pay for a quorum read.

By the end: **A Raft implementation in Go that elects a leader, replicates a log, and stays safe across a partition and a rolling restart.**

3 Systems That Stay Up

Build a service that keeps its data correct under duplicate delivery, rebalancing, and overload.

6 sessions
$108 alone

3.1	**Exactly-Once Is a Fiction**	At-least-once delivery plus idempotency keys, and what Kafka's transactional producer does not cover.
3.2	**Sagas, Not Two-Phase Commit**	Why 2PC blocks on a dead coordinator, and writing compensations you can safely run twice.
3.3	**The Log as Truth**	Kafka 4.3 on KRaft: partitions, offsets, consumer groups, and ordering that only holds per key.
3.4	**Sharding and Rebalancing**	Consistent hashing versus ranges, hot keys, and moving a shard without losing an in-flight write.
3.5	**Retries Make It Worse**	Retry amplification, thundering herd, backoff with jitter, and a retry budget that caps the blast.
3.6	**Backpressure and Load Shedding**	Little's Law, bounded queues, admission control, and why an autoscaler is not backpressure.

By the end: **A sharded order service on Kubernetes that keeps balances correct through a duplicate flood, a shard move, and a retry storm.**

4 Prove It Breaks

See inside a running distributed system, then attack it until it tells you something you did not know.

5 sessions
$108 alone

4.1	**Trace Across Six Hops**	OpenTelemetry context propagation, W3C traceparent, and sampling that still keeps the rare failure.
4.2	**Tail Latency and SLOs**	Why the mean hides the outage, tail amplification across fan-out, and one SLO worth paging on.
4.3	**Break the Cluster**	Chaos Mesh 2.8 against your own Raft cluster: pod kill, partition, packet loss, and clock skew.
4.4	**Jepsen Your Own Store**	Generate a history under fault, then let Elle find the stale read your unit tests never saw.
4.5	**Deterministic Simulation Testing**	Seeded clock, network, and scheduler, replaying one heisenbug exactly, as etcd now does with Antithesis.

By the end: **A fault-injection report: five faults driven into a live cluster, each with the trace showing blast radius and one measured fix.**

Tools you will use

Kubernetes 1.36 | etcd 3.7 | Go 1.25 | Apache Kafka 4.3 (KRaft) | Temporal | OpenTelemetry Collector 0.154 | Prometheus 3.13 LTS | Grafana Tempo | Chaos Mesh 2.8 | Jepsen 0.3.10 with Elle

What you will build

1. **Raft From the Paper**
   Election and log replication that survive a partition
2. **The Order Service That Holds**
   Sharded, idempotent, and correct under a retry storm
3. **Break Your Own Cluster**
   Five injected faults, five traces, one measured fix

Registration

Phase 1: Partial Failure and Time	$108
Phase 2: Replication and Consensus	$108
Phase 3: Systems That Stay Up	$108
Phase 4: Prove It Breaks	$108
Whole track, all 4 phases	**$360**

Buying every phase separately costs $432, so the whole track saves $72.

No payment is taken online. Registering creates an order and payment instructions are sent with it. Access opens once payment is confirmed.