Chaos Engineering, Flamegraphs & eBPF Observability

SRE Team Demonstration - Key Takeaways

What We Demonstrated Today

1. Flamegraph Profiling - Visual Performance Analysis

- **Tool**: Go pprof + FlameGraph
- **What it does**: Shows exactly where CPU time is spent
- **Key finding**: 40% of CPU in database operations, 12% in JSON serialization
- **Value**: Identifies optimization opportunities without code instrumentation

2. Chaos Engineering: Network Partition - Failure Injection

- **Tool**: iptables + eBPF tracing
- **What it does**: Simulates network failure between services
- **Key finding**: 5.8 second recovery time, 0 orders lost, retry logic validated
- **Value**: Test resilience before production incidents

3. Chaos Engineering: CPU Throttling - Resource Pressure

- **Tool**: Docker resource limits + eBPF scheduler tracing
- **What it does**: Limits CPU to simulate overload
- **Key finding**: 10x latency increase, 0% errors, graceful degradation confirmed
- **Value**: Measure system capacity and scaling thresholds

Business Impact

Capability	Investment	ROI	Impact
Chaos Engineering	1 week setup	6 months	Reduce MTTR 60%, find issues pre-production
Flamegraph Profiling	1 day setup	3 months	Reduce cloud costs 30-40% via optimization

Capability	Investment	ROI	Impact
eBPF Observability	2 days training	Immediate	Debug production with <1% overhead

Total Investment: ~2 weeks team time **Cost**: $0 (all open-source tools) **Expected Annual Savings**: $50K-$200K (via incident reduction + optimization)

Key Technical Insights

Flamegraphs Revealed

```
CPU Time Distribution:
  Database queries:        40% ← Optimization target
  JSON serialization:      12% ← Consider binary protocol
  Order validation:        15% ← Can parallelize
  Network I/O:              8%
  Other:                   25%
```

Network Chaos Results

```
Timeline:
  0s:    Network partition injected (iptables DROP)
  <1s:   First connection failure detected
  1-25s: TCP retransmissions visible via eBPF
  25s:   Connection timeout (kernel default)
  90s:   Partition healed
  91.2s: First successful connection
  95.8s: Full operational recovery

Data Loss: 0 orders (queued during outage)
```

CPU Throttle Results

```
CPU Limited to 10% (from 100%):
  Latency:      120ms → 1150ms (10x increase)
  Queue depth:  0 → 51 requests
  Error rate:   0% (graceful degradation)
  Recovery:     20 seconds to process backlog

Capacity Finding: System handles 3x current load before degradation
```

---

eBPF - What It Showed Us

Without modifying application code, we traced:

TCP connection attempts and retransmissions Kernel scheduler delays and context switches Connection timeout behavior (25s default) Process wait time in CPU runqueue (850ms avg under pressure)

Performance Impact: <1% CPU overhead **Production Safe**: Yes - kernel-level, read-only

---

Make Targets - Try It Yourself

Profiling

```
make profile-cpu          # Generate CPU flamegraph (30s)
make profile-all          # All profile types (CPU, heap, mutex, etc.)
make profile-view         # Open flamegraphs in browser
make profile-analyze      # AI analysis of performance
```

Chaos Experiments (10 Total)

```
make chaos-network-partition  # eBPF network tracing
make chaos-cpu-throttle       # Scheduler performance
make chaos-oom-kill           # Memory pressure
make chaos-disk-io            # I/O starvation (advanced eBPF)
make chaos-cascade            # Multi-service failure
make chaos-flamegraph         # Profiling during chaos
```

Utilities

```
./generate-load.sh 60 30  # Generate load (60s, 30 req/s)
make help                 # Show all targets
```

Tools We Used (All Free/Open Source)

Tool	Purpose	URL
bpftrace	eBPF tracing	https://github.com/iovisor/bpftrace
pprof	Go profiling	https://pkg.go.dev/net/http/pprof
FlameGraph	Visualization	https://github.com/brendangregg/FlameGraph
Prometheus	Metrics	https://prometheus.io
Grafana	Dashboards	https://grafana.com
Docker	Containers	https://docker.com

Industry Adoption

Companies using these techniques: - **Netflix**: Chaos Monkey (pioneered chaos engineering) - **Google**: Continuous profiling in production - **Amazon**: eBPF for production observability - **Facebook**: Profiling at scale (millions of servers) - **Uber**: Chaos testing for ride reliability

Conference presentations: - SREcon (annual chaos engineering track) - KubeCon (eBPF observability sessions) - QCon (performance engineering talks)

Next Steps for Our Team

Immediate (This Week)

- ☐ Run remaining 7 chaos experiments in staging
- ☐ Review flamegraphs for optimization opportunities
- ☐ Document baseline performance metrics

Short Term (This Month)

- ☐ Identify 3 production optimizations from profiling
- ☐ Create runbooks based on chaos findings
- ☐ Team training: bpftrace basics (4 hour workshop)
- ☐ Set up Grafana alerts based on chaos thresholds

Long Term (This Quarter)

- ☐ Integrate chaos tests into CI/CD pipeline
- ☐ Continuous profiling in staging environment
- ☐ eBPF production debugging playbooks
- ☐ Performance budgets and SLO definitions

Resources & Documentation

Internal Docs

- **Complete Demo Guide**: `docs/SRE_DEMO_GUIDE.md` (45 pages)
- **Quick Cheatsheet**: `docs/SRE_DEMO_CHEATSHEET.md` (5 pages)
- **Flamegraph Guide**: `docs/FLAMEGRAPH_GUIDE.md`
- **Chaos Experiments**: `chaos-experiments/README.md` (10 experiments)

External Resources

- **Brendan Gregg's eBPF Book**: http://www.brendangregg.com/bpf-performance-tools-book.html
- **Principles of Chaos**: https://principlesofchaos.org/

- **SRE Book (Google)**: https://sre.google/books/
- **pprof Tutorial**: https://go.dev/blog/pprof

Training

- **eBPF Tutorial**: https://github.com/iovisor/bpf-docs
- **Flamegraph Examples**: http://www.brendangregg.com/flamegraphs.html
- **Chaos Eng Workshop**: https://principlesofchaos.org/workshops/

Demo Artifacts Available

CPU flamegraph visualizations (SVG files) All profile types (heap, goroutine, mutex, allocs, block) Chaos experiment detailed reports eBPF trace samples showing kernel behavior Grafana dashboard screenshots (before/during/after) Performance baseline measurements

Access: Shared drive folder or contact [demo presenter]

FAQs

Q: Can we run this in production? A: Profiling and light chaos (CPU throttle) - yes. Network partition and OOM kill - staging only.

Q: What's the learning curve? A: Basic flamegraph reading: 1 hour. eBPF basics: 1 day. Advanced chaos engineering: 1 week.

Q: Do we need to buy anything? A: No. All tools are free and open source.

Q: What about non-Go services? A: Flamegraphs work with Java, Python, C++, Rust, etc. eBPF is language-agnostic.

Q: How do we start? A: Start with profiling (lowest risk, highest immediate value). Add chaos tests in staging. Scale from there.

Q: What if something breaks during chaos? A: All experiments include automatic cleanup. Services auto-recover via Docker restart policies.

Success Metrics We'll Track

Engineering Metrics: - Mean Time To Resolution (MTTR): Target 60% reduction - Pre-production bug detection: Target 80% of issues found in staging - Performance optimization wins: Track cost savings

Business Metrics: - Infrastructure costs: Target 30-40% reduction via optimization - Incident count: Track reduction in production issues - Customer impact: Measure reduction in user-facing outages

Adoption Metrics: - Team proficiency: Quarterly skills assessment - Tool usage: Weekly profiling runs, monthly chaos tests - Runbook updates: Chaos findings → operational procedures

---

Contact & Follow-Up

Demo Presenter: [Your name/email] **Team Lead**: [Team lead name/email] **Slack Channel**: #sre-observability (proposed)

Scheduled Follow-Ups: - Team workshop: [Date TBD] - Review session: [Date TBD] - Quarterly review: [Date TBD]

Key Quotes from Today

> "40% of our CPU time is in database operations - that's our biggest optimization opportunity"

> "5.8 seconds from network failure to full recovery - our retry logic is working correctly"

> "The system degraded gracefully under CPU pressure - slow, not broken"

> "eBPF showed us kernel behavior we couldn't see from application logs"

Thank you for attending! Questions? Let's discuss.

Document prepared: November 2025 Technologies: bpftrace, pprof, FlameGraph, Prometheus, Grafana, Docker Demo environment: simulated_exchange (Go microservices)