The background features a dark grey color with white and light blue circuit board patterns in the corners. These patterns consist of various lines, circles, and dots, resembling a printed circuit board (PCB) layout. The patterns are most prominent in the top-left, top-right, and bottom-right corners, with some extending into the bottom-left corner.

# Open Source

# Standing on the shoulders of giants

A Practical Guide to Open Source in Development & Production

# Today's Journey

- The 'Free' Software Story: How we got here
- Open Source You're Already Using (surprise!)
- License Types: The Good, The Bad, and The Lawyer-Approved
- Development Environments: Your New Best Friends
- Production Deployments: Where the Magic Happens

# Once Upon a Time in Computing...

- 1960s-70s: Software was free! Everyone shared code like recipes
- 1980s: Companies realized 'Hey, we can charge for this!'
- 1983: Richard Stallman says 'Not on my watch!' - GNU Project born
- 1991: A Finnish student named Linus emails: 'I'm doing a (free) OS...'
- The rest, as they say, is history (and lawsuits, and innovation!)

# Two Development Philosophies

- The Cathedral: Top-down, controlled, 'we know best' (traditional software)
- The Bazaar: Chaotic, collaborative, 'given enough eyeballs, all bugs are shallow'
- Eric Raymond's 1997 essay changed how we think about development
- Turns out, the bazaar model works surprisingly well!
- Modern open source development is basically one big, global bazaar

# The Secret Sauce of Success

- Economics: Free is a very compelling price point
- Innovation: Thousands of brains beat dozens every time
- Trust: You can actually see what the code does (no secret backdoors)
- Flexibility: Don't like something? Fork it and make your own!
- Community: Never code alone again (for better or worse)

The background features a dark grey gradient with white circuit board patterns in the corners. The top-left and bottom-right corners have more complex, dense patterns, while the top-right and bottom-left corners have simpler, more linear patterns. The main text is centered in the middle of the page.

# Open Source All Around You

Spoiler: You're Already an Open Source User!

# That Device in Your Pocket

- Android: Built on Linux kernel (sorry iPhone users, you're partly open too)
- 70% of smartphone users run on open source daily
- Your iPhone's Safari? WebKit engine is open source
- Most mobile apps use open source libraries under the hood
- Even your phone's WiFi and Bluetooth stacks are open source

# How You're Seeing This Presentation

- Apache & NGINX: Power 60%+ of all websites you visit
- Linux: Runs 90% of the cloud, 100% of the top supercomputers
- OpenSSL: Keeps your passwords safe (when not heartbleeding)
- DNS servers: Mostly BIND (open source since 1984)
- Email infrastructure: Postfix, Sendmail, Dovecot keeping spam flowing

# Tools of Your Trade

- VS Code: Microsoft's gift to developers (yes, really!)
- Git: Because 'final\_final\_v3\_actualfinal.doc' doesn't cut it
- Node.js: JavaScript everywhere (whether you like it or not)
- React, Vue, Angular: The holy trinity of front-end frameworks
- Python, PHP, Ruby: The languages running half the web

# Your Netflix & Chill is Open Source

- VLC: The media player that plays literally everything
- Netflix backend: Runs on open source (FreeBSD, Linux, etc.)
- Spotify, YouTube, Facebook: All heavily rely on open source
- Your smart TV: Probably running Linux or Android TV
- Gaming: Steam, Unreal Engine, even Minecraft - open source components

# Even 'Proprietary' Software Uses Open Source

- Microsoft: Now one of the largest open source contributors (!)
- Apple: WebKit, Swift, and tons of BSD code
- Google: Chrome, Android, TensorFlow - the list goes on
- AWS, Azure, Google Cloud: Built on Linux and open source tools
- Your proprietary app: Probably uses 100+ open source libraries



# Understanding Open Source Licenses

The 'Free' in 'Free Software' Needs Some Explaining



# Two Kinds of 'Free'

- Free Beer: No cost, but that's just a happy accident
- Free Speech: Freedom to use, modify, share, and distribute
- Open source focuses on practical benefits and development model
- Free software (GNU) focuses on user freedom and ethics
- Same software, different philosophy (like Coke vs. Pepsi but nerdier)

# Permissive Licenses: 'Do Whatever, Just Credit Us'

- MIT License: The 'I trust you' license (most popular on GitHub)
- Apache 2.0: MIT's lawyer-friendly cousin (with patent protection)
- BSD: The OG permissive license (FreeBSD, OpenBSD)
- Key trait: Use in proprietary software? Go for it!
- Catch: You can take the code but can't sue for patent issues

# Copyleft: 'Share and Share Alike'

- GPL (General Public License): The 'viral' license Stallman loves
- LGPL: GPL's more permissive younger sibling (for libraries)
- AGPL: GPL but closes the 'web service loophole'
- Key trait: If you modify and distribute, you **MUST** share source code
- Philosophy: Freedom for users, not just developers

## Summary Table

License	Copyleft Strength	Trigger	Allows Proprietary Linking?	Web Service Obligation?
GPL	Strong	Distribution (binary/source)	✗ No	✗ No
AGPL	Strongest	Distribution <b>and</b> network use	✗ No	✓ Yes
LGPL	Weak	Distribution (of library)	✓ Yes	✗ No

## In Plain Terms

- **GPL:** "If you share me, share your changes too."
- **AGPL:** "If you let people *use* me over a network, share your changes too."
- **LGPL:** "You can use me in your app (even closed-source), but if you change *me*, share those changes."

# Where Licenses Fall on the Freedom Scale

- Public Domain (CC0): 'Take it, it's yours, I don't care'
- Permissive (MIT, Apache): 'Take it, just say thanks'
- Weak Copyleft (LGPL): 'Take it, share changes to MY code'
- Strong Copyleft (GPL): 'Take it, share EVERYTHING'
- AGPL: 'Take it, share everything, even on servers'

# Mixing Licenses: A Recipe for Confusion

- Problem: GPL + Apache in same project = potential legal issues
- Solution: Use license-compatible components or dual-license
- Tools: FOSSA, Black Duck, WhiteSource scan dependencies
- Reality: Most companies have a 'banned license' list
- Pro tip: Keep a software Bill of Materials (SBOM)

# Don't Be 'That Person'

- Mistake 1: Removing copyright notices ('I made this!'... no you didn't)
- Mistake 2: Using GPL code in proprietary software (lawyers will find out)
- Mistake 3: Violating attribution requirements (credits matter!)
- Mistake 4: Assuming 'no license' means 'free to use' (it doesn't!)
- Mistake 5: Not checking licenses of dependencies (it's turtles all the way down)

The background features a dark grey color with white circuit board patterns in the corners. These patterns consist of various lines, curves, and small circles, resembling a printed circuit board (PCB) layout. The patterns are most prominent in the top-left, top-right, and bottom-right corners, with some extending towards the center.

# Open Source Development Environments

Where the Magic Happens (Before Production Breaks)

# The Development Case for Open Source

- Transparency: Can inspect code for bugs/security issues
- No vendor lock-in: Can fork if vendor direction changes
- Community: Shared knowledge, plugins, ecosystem
- Interoperability: Often better standards compliance
- Longevity: Won't disappear if company folds

# Linux: Not Just for Servers Anymore

- Ubuntu Desktop: Windows, but for developers who like terminal
- WSL 2: Linux on Windows (because Microsoft gave up fighting)
- Package managers: apt, snap - install anything with one command
- Developer tools: Built-in compilers, interpreters, everything
- Bonus: Feel like a hacker in movies with a terminal window

# Docker: 'It Works on My Machine'

- Package entire app + dependencies in a container (like a ZIP on steroids)
- Development: Everyone uses same environment, no more setup hell
- Testing: Spin up databases, services in seconds
- Sharing: 'docker run' and you're up and running
- Learning curve: 5 commands get you 80% there

# Git: Time Travel for Code

- Track every change, never lose code again (unless you force push)
- Branching: Try crazy ideas without breaking main code
- Collaboration: Multiple developers, no stepping on toes (much)
- GitHub/GitLab/Bitbucket: Your code's social network
- Confession: Everyone googles Git commands daily

# Popular Open Source Dev Stacks

- LAMP: Linux + Apache + MySQL + PHP (the classic)
- MEAN: MongoDB + Express + Angular + Node.js (the JavaScript takeover)
- JAMstack: JavaScript + APIs + Markup (the new hotness)
- Your custom stack: Mix and match what works for YOU
- Key point: All free, all proven, all community-supported

# Your Open Source Toolbox

- IDEs: VS Code, Vim, Emacs (pick your religion)
- Build tools: Maven, Gradle, npm, webpack (because compilation is hard)
- Testing: Jest, pytest, JUnit (test or be tested in production)
- Debugging: gdb, Chrome DevTools (when console.log isn't enough)
- Documentation: Sphinx, Jekyll (for when you actually document things)

The background features a dark grey gradient with white circuit board traces and circular components in the corners. The top-left and bottom-right corners have more complex, dense circuit patterns, while the top-right and bottom-left corners have simpler, more linear traces.

# Open Source in Production

Where Your Code Meets Reality (and Users)

# Development vs. Production: A Tale of Two Worlds

- Development: Fast and loose, mistakes are learning opportunities
- Production: Slow and careful, mistakes are career opportunities
- Development: 'It works!' is success
- Production: 'It works under load, securely, 24/7' is the baseline
- Open source powers both, but production needs extra love

# Linux in Production: The Obvious Choice

- Ubuntu Server: Popular, LTS versions, 5-year support
- RHEL/Rocky/Alma: Enterprise support, certified for everything
- Debian: Ultra-stable, if boring is your thing (it should be)
- Why Linux: Stability, security, performance, cost
- Fun fact: Even Microsoft Azure runs more Linux than Windows

# Managing Containers at Scale

- Docker Swarm: Simple, built-in, 'good enough' for many
- Kubernetes: Complex, powerful, 'cloud native' buzzword certified
- Portainer: Web UI for mortals who don't memorize kubectl commands
- Reality: Start simple, scale complexity as needed
- Warning: Don't Kubernetes your 3-user app (yes, we see you)

# The Front Door to Your Applications

- NGINX: Fast, efficient, handles 10k+ concurrent connections
- Apache: The veteran, module for everything, slightly slower
- Caddy: Automatic HTTPS, config is actually readable
- HAProxy: Load balancing beast, Swiss Army knife of proxies
- Role: SSL termination, load balancing, caching, security

# Data: The Important Stuff

- PostgreSQL: Feature-rich, ACID-compliant, 'most advanced open source DB'
- MySQL/MariaDB: Fast, popular, powers WordPress (and millions more)
- MongoDB: Document store, when schemas are more like suggestions
- Redis: In-memory cache, session store, pub/sub, Swiss Army data tool
- Critical: Backups, replication, monitoring - data loss is not an option

# You Can't Fix What You Can't See


- Prometheus + Grafana: Metrics and pretty dashboards
- ELK Stack: Logs aggregation and search (find that error!)
- Jaeger/Zipkin: Distributed tracing for microservices
- Uptime monitoring: Is it up? (The million-dollar question)
- Philosophy: Monitor everything, alert on what matters

# Automating the Deploy Button

- Jenkins: The Swiss Army knife (been around forever)
- GitLab CI: Git and CI/CD in one platform
- GitHub Actions: Cloud-native, YAML-configured workflows
- Goal: Code commit → automated tests → deploy (while you sleep)
- Reality: More time debugging YAML than actual code

# Security: Not an Afterthought

- Updates: Automate security patches (with testing!)
- Scanning: Check for vulnerabilities in dependencies
- Secrets management: HashiCorp Vault, not hardcoded passwords
- Network security: Firewalls, VPNs, principle of least privilege
- Reality: Open source is auditable, but YOU have to do the auditing

The background features a dark grey gradient with white circuit board patterns in the corners. These patterns consist of various lines, curves, and small circles, resembling a printed circuit board (PCB) layout. The patterns are most prominent in the top-left, top-right, and bottom-right corners, with some extending into the bottom-left corner.

# Making It Work: Practical Strategies

From 'Sounds Nice' to 'Actually Doing It'

# First Steps (Without Overwhelming Yourself)

- Step 1: Audit what you're already using (surprise yourself)
- Step 2: Pick ONE thing to improve/migrate first
- Step 3: Build internal knowledge (training, docs, champions)
- Step 4: Start small, prove value, then scale
- Step 5: Celebrate wins, learn from failures (there will be some)

# The Eternal Question

- Build: When you need something unique, have time and talent
- Buy: When time-to-market matters, need support/guarantees
- Borrow (open source): When the problem is solved, community exists
- Reality: Most apps are 80% borrowed, 15% bought, 5% built
- Smart move: Don't reinvent wheels, focus on your unique value

# Being a Good Open Source Citizen

- Use: Download, try, provide feedback
- Contribute: Bug reports, documentation, actual code
- Sponsor: Fund projects you depend on (they need coffee too)
- Share: Blog, speak, teach others what you learned
- Golden rule: Take from the commons, give back to the commons

# Learn From Others' Mistakes

- Pitfall 1: Picking tools because they're 'cool' (resume-driven development)
- Pitfall 2: Underestimating operational complexity (Kubernetes for a blog?)
- Pitfall 3: Ignoring licenses (legal says no, but you did it anyway)
- Pitfall 4: No succession plan (one developer knows everything)
- Pitfall 5: Not budgeting for maintenance (open source isn't 'set and forget')

# Open Source + Progress: A Practical Approach

- Containerize: Package Progress apps with Docker for consistency
- Orchestrate: Manage deployments with Portainer or simpler tools
- Monitor: Add Prometheus/Grafana for observability
- Automate: CI/CD for Progress code deployments
- Integrate: Use open APIs and message queues to connect systems

# Moving to Open Source: The Gradual Approach

- Phase 1: Infrastructure (dev environments, tools)
- Phase 2: Surrounding services (web servers, databases, caching)
- Phase 3: Containerization (package existing apps)
- Phase 4: Orchestration (manage containers at scale)
- Phase 5: Modernization (microservices, APIs, new features)

# How to Know It's Working

- Development velocity: Faster feature delivery
- Deployment frequency: Daily/weekly instead of monthly/quarterly
- Time to recovery: Minutes/hours, not days
- Cost reduction: Lower licensing and infrastructure costs
- Developer happiness: Recruitment, retention, satisfaction surveys

The background features a dark grey gradient with white circuit board patterns in the corners. The top-left and bottom-right corners have more complex, dense patterns, while the top-right and bottom-left corners have simpler, more linear patterns.

# The Future of Open Source

Where Do We Go From Here?

# What's Hot Right Now

- AI/ML: TensorFlow, PyTorch democratizing artificial intelligence
- Cloud Native: Everything Kubernetes, service mesh, serverless
- Security: Supply chain security, SBOM, automated scanning
- Sustainability: Open source for green computing
- Collaboration: GitHub Copilot, AI pair programming (open source meets AI)

# Convincing the Decision Makers

- Cost: 'Free' is always a good starting point for budget discussions
- Risk: Proven, battle-tested by tech giants (safer than proprietary)
- Talent: Developers want to work with modern, open source tools
- Flexibility: No vendor lock-in, switch when needed
- Innovation: Stay competitive with industry standard tools

# It's Not All Sunshine and Rainbows

- Sustainability: Many projects rely on unpaid volunteers (burnout risk)
- Security: Supply chain attacks, malicious dependencies
- Complexity: Choice paralysis with 1000 tools for every problem
- Support: Community help is great until you need it at 3 AM
- Cultural: Organizational resistance to 'free stuff'

# How to Give Back to the Community

- Code contributions: Fix bugs, add features you need
- Documentation: Write guides, improve README files
- Financial support: Sponsor maintainers, buy support contracts
- Advocacy: Speak at conferences, write blogs, teach
- Be kind: Encourage beginners, help in forums, positive vibes

# Remember These Points

- You're already using open source (probably more than you think)
- Licenses matter - understand them before using code
- Start small, prove value, then scale adoption
- Open source isn't just free, it's about community and collaboration
- Give back - the commons only works if we all contribute

# Where to Go From Here

- Try it: Install Linux, run Docker, deploy a container
- Read: GitHub trending, Hacker News, dev.to, blogs
- Watch: Conference talks, YouTube tutorials, live coding
- Practice: Contribute to projects, start your own
- Connect: Join communities, attend meetups, make friends

The background features a dark grey gradient with white and light blue circuit board traces and components. These traces are most prominent in the corners, forming a frame-like structure. The central text is rendered in a clean, sans-serif font.

# Questions & Discussion

Let's Talk About Open Source!

# Thank You!

- Remember: Open source is a journey, not a destination
- Stay curious, keep experimenting, and have fun
- The best way to learn is by doing (and breaking things)
- Community is your greatest asset - use it!
- Now go forth and open source all the things!  
(Responsibly)