Conference Reports

In this issue:

72 LISA14

75 Advanced Topics Workshop at LISA14

LISA14

November 9–14, 2014, Seattle, WA

Summarized by Herry Herry and Josh Simon

Invited Talks

Making "Push on Green" a Reality: Issues & Actions Involved in Maintaining a Production Service

Dan Klein, Google, Inc.

Summarized by Josh Simon (jss@clock.org)

Despite encouragement from the audience, Dan didn't give the talk as an interpretive dance. The talk basically asked (and answered) the question, "What is 'push on green,' and what do you need to do before you can get to it?"

He laid out his assumptions for this talk: you have at least one server, at least one environment, some number of dependents or dependencies, the need to make updates, and a limited tolerance for failure.

What's "push on green"? When something—such as a human, build system, or test suite—says it's okay to release something, do it. It's unfortunately complicated by reality: how do you avoid rollout pain with a new or modified service (or API or library or...)? In summary:

- **Developing**. Peer reviewed code changes; *nobody* does a check-in-and-push (with possible exceptions when Production is broken, but the code review needs to happen after the fact). Is the code readable? Well-documented? Test your code—with both expected and unexpected conditions. Does it fail gracefully? Use new libraries, modules, and APIs; don't do a "first upgrade in five years" thing.
- **Testing**. Unit tests, module tests, end-to-end tests, smoke tests and probers, and regression tests. Find a bug? Write a test to reproduce it, patch it, and rerun the test. (Example: OpenSSL has only five simple tests at a high level and hundreds of modules that aren't directly tested at all.)
- **Monitoring**. Volume (how many hits), latency, throughput (mean, minimum, maximum, standard deviation, rate of change, and so on); historical data and graphing; alerting and service level agreements (SLAs). As a side note, SLAs require service level objectives (SLOs), which require service level indicators (SLIs).
- **Updating (and rolling back)**. Should be automated and mechanical idempotent processes. This requires static builds, ideally with human-readable version numbers like *yyyymmdd_rcn*. It needs to be correlated with monitoring. You can mark a version as "live," and then push is just changing the pointer to that live version; rollback is re-marking the "old" version as live and updating the pointer ("rolling forward to a previous version"—assuming no database schema changes anyhow). You should also have canary jobs; a canary job is in the case when you have more than one machine or process. You say "some amount of traffic will hit the canary job with the new version." You need to check the canary first to see whether it crashed. If you monitor the canaries and let them have some fraction of the traffic, you can look at those graphs and check for anomalies and trending and see whether the canary works as expected. If it looks good, you can push things live. If it doesn't, only a small fraction of users are affected for a short period of time.

Your organization needs a cross-cultural mindset across all of these.

So how do you do a safe rollout? In general:

- Silence the relevant alerts in your monitoring system.
- Update the canary jobs.
- Run your smoke tests.

- Let canaries "soak," or run for a while; the code or test might require some number of iterations such as loading a disk cache.
- Push the remaining jobs.
- Run the smoke tests again.
- Unsilence alerts.

What about making the configuration changes? You can have job restarts with runtime flags, or HUPping the job to reread the config file. The latter is faster but riskier.

In about 84 weeks with this process, Google went from roughly five rollouts per week to upwards of 60 (with a peak of 75) *and* freed up an FTE engineer. Having more frequent and smaller rollouts helps developers, who don't have to wait for the "weekly build" to release their code.

Process improvements they've made include automated recurring rollouts (using a 4.5-day week, excluding weekends and Friday afternoons), inter-rollout locking (to prevent stomping on each other), schedules of rollouts (to prevent things from happening when people are sleeping), and one-button rollbacks.

Future enhancements to the process include rollback feasibility (how easy is it to roll back the release, e.g., if there are schema changes?), continuous delivery (just release it automatically if there's a change in the binary or config file checked in), rollout quotas (prevent someone from taking all the slots for a person, team, or tool), and green on green (if there's continuous delivery and something breaks, should that halt additional deployments?).

This is an evolutionary process. Things *will* go wrong—things break, and so we need to adjust attitudes. Find the reasons why something went wrong, not to assign blame but to fix the process. Let humans be smart and machines be repeatable. Have a Big Red Button so a human can stop things if needed.

You don't need to be at Google-scale to do this. And, sorry, there are no silver bullets. It's a laborious process with lots of baby steps. You have to be careful and not take shortcuts, just keep going.

While waiting for the Q&A to start, Dan did perform an interpretive dance after all.

For more information, please see the "Push on Green" article in the October 2014 (vol. 39 no. 5) issue of *;login:*.

Distributing Software in a Massively Parallel Environment

Dinah McNutt, Google, Inc.
Summarized by Josh Simon (jss@clock.org)

Dinah McNutt has been coming to LISA since LISA IV (1990) and chaired LISA VIII (1994), and while she used to be a sysadmin she's now a release engineer. One of her passions is packaging; she's fascinated by different package managers and she talked about Google's.

The problem is that with very large networks, it may take a long time to distribute things; there are bottlenecks (such as network, disk, CPU, and memory), a machine may be offline, networks might be partitioned ("you can't get there from here"), and there are even concurrent writers.

Google's package management system is called Midas Package Manager (MPM). The package metadata (more below) is stored in their Bigtable Database, and package data is stored in their Colossus File System and replicated. The transport mechanism is a custom P2P mechanism based on torrent technology.

An MPM package and metadata contain the contents of the package (the files), a secure hash of the unique version ID, signatures for verification and auditing, labels (such as "canary," "live," "rc" with date, and "production" with date; for more on the "canary" and "live" labels see the preceding talk), pre-packaging commands, and optionally any pre- and post-installation commands.

She gave a quick case study: a config file needs to go to thousands of machines, so the relevant pieces are packaged into an MPM file, and a job (that is, a process running in a container) on each remote machine fetches and installs a new version of that MPM every 10 minutes, so the config changes can go out quickly. A post-fetch script is in the MPM to install the new config file. Easy, right?

Alas, it's not quite that simple: machines may be offline, bottlenecks must be minimized, jobs have to specify the version of a package, jobs on the same machine may use different versions of the same package, the system must be able to guarantee files aren't tampered with in flight, and the system must be able to roll back to a previous version.

At package creation, the build system creates a package definition file, which includes the file list; ownership and permissions; pre- and post-install and remove commands; and all is generated automatically. Then it runs the build command. It can apply labels and signatures at any point during the process.

If files going into the package aren't changed, a new package isn't created; the label or signature is just applied to the existing (unchanged) package.

The metadata can be both immutable (who, when, and how it was built; list of files, attributes, and checksums; some labels, especially those with equals signs; and version ID) and mutable (labels without equals signs, and cleanup policy).

The durability of an MPM package depends on its use case: test packages are kept for only three days, ephemeral packages are kept for a week (usually for frequently pushed configuration files), and durable packages are kept for three months after their last use (and stored only on tape thereafter).

Distribution is via pull (by the client-side job), which avoids network congestion (things are only fetched when needed) and

lets the job owners decide when to accept new versions (e.g., "wait until idle"). But since job owners can decide when to accept new versions, there either has to be extra logic in the job to check for new versions *or* the ability to restart jobs easily, and it can be difficult to tell who's going to be using a specific version.

The package metadata is pushed to Bigtable (which is replicated) immediately. Root servers read and cache data from their local Bigtable replica. MPM queries the local root server; failover logic is in the client, so if requests fail they're automatically redirected to another Bigtable replica.

Package data is in the Colossus File System, scattered geographically. It's a two-tiered architecture; frequently used packages are cached "nearby" (closer to the job). The fetch is via a torrent-like protocol, and the data is stored locally; so as long as it's in use you don't need to talk to either Bigtable or Colossus. There's only one copy on the machine no matter how many jobs on the machine use it. They have millions of fetches and petabytes of data moving daily.

Security is controlled via ACLs. Package namespace is hierarchical, like storage/client, storage/client/config, and storage/server. ACLs are inherited (or not). There are three levels of access:

- **Owner** can create and delete packages, modify labels, and manage ACLs.
- **Builder** can create packages and add/modify labels.
- **Label** can control who can add/modify specific labels: production.*, canary, my_label=blah, and so on.

Individual files can be encrypted within a package, and ACLs define who can decrypt the files (MPM can't). Encryption and decryption are performed locally and automatically, which allows for passwords that aren't ever stored unencrypted.

Signatures can be signed at build time or later. Secure key escrow uses the package name and metadata so a package can be verified using the name and signer.

Why love MPM? There's an mpmdiff that can compare any two packages regardless of name (like the file owner, file mode, file size, file checksums, and the pre- and post-scripts).

Labels are great. You can fetch packages using labels. You can use them to indicate where the package is in the release process (dev, canary, or production). You can promote a package by moving labels from one package to another, although some labels (those with equals signs) are immutable and can't be moved. Some labels are special ("latest," which shouldn't be used because that bypasses using a canary). They can assist in rollbacks (like "last_known_good" or "rollback" to label the current MPM while promoting the new one).

There's a concept of file groups: it's a grouping of binaries within an MPM. Binaries can belong to more than one group. Common practice is to store both stripped and unstripped binaries in the same MPM but in different file groups, to ensure the unstripped and stripped binaries match when troubleshooting problems.

There's a Web interface to browse all MPMs and show the metadata. It also shows graphs by size (so you can see how file groups change over time).

In the Q&A, Dinah addressed various questions.

Because job owners have control over when they accept new versions, the MPM team can't guarantee that every machine in production runs the "correct" version; you may have to nag people to death to upgrade. The release processes can therefore vary wildly. The SREs are good and well-respected; they're gatekeepers to keep the release processes sane. The automated build system (which is optional) enforces workflows. There is a continuous testing system where every command line submitted triggers a test. They insist that formal releases also run tests since the flags are different.

One thing possibly missing is dependency management, but that's because packages are self-contained. Performing a fetch pulls in the dependent packages, and the code explicitly lists the dependencies. In MPM, the goal was to avoid dependency management since anything can be in a package.

Building a One-Time-Password Token Authentication Infrastructure

Jonathan Hanks, LIGO Lab/California Institute of Technology; Abe Singer, Laser Interferometer Gravitational Wave Observatory, Caltech
Summarized by Herry Herry (h.herry@sms.ed.ac.uk)

Jonathan Hanks began by saying that his team built a one-time-password token authentication infrastructure, called LIGO, in part because they want to prevent credential theft before it is too late. His organization produced several requirements of the token-based solution: one token to rule them all; the token must be produced by a physical device; trust no one; and the system must be distributed, fault-tolerant, use open standards, and be cheap.

After considering the requirements, his team decided to build a custom semi-distributed system where some sites run authentication servers while others do not. The authentication system can split and join depending on circumstances. The system is using MIT Kerberos with a custom user database. The server periodically synchronizes its data with others to replicate the user database.

All services are connected to the authentication system using PAM. Each user of a service must employ a YubiKey device to generate a one-time password during authentication. The device itself is sent simply by mail, and the user must activate the device before it can be used.

Jonathan closed his talk by saying that it is important to own the system by themselves because there is no secret in the system. Thus, they can fully trust it.

JEA—A PowerShell Toolkit to Secure a Post-Snowden World

Jeffrey P. Snover, Microsoft

Summarized by Herry Herry (h.herry@sms.ed.ac.uk)

Jeffrey Snover began by illustrating that Edward Snowden is more powerful than General Michael Hayden (former CIA director), mainly because he held "the key to the kingdom": there was no limitation on what he could do on the system as a system administrator. This issue motivated Jeffrey's team to build JEA (Just Enough Admin). But although we should manage the risk, we cannot eliminate it.

Snover explained that with JEA, we can prescribe which actions that can be executed by particular admins. With this, people do not need admin privileges to do their job. He also added that all admin actions got logged, which is very useful for auditing. One of the attendees described JEA with the precise term "firewall shell."

Jeffrey mentioned that JEA is integrated inside the PowerShell. He said that we can define "command visibility" data, which drives the parsing capability of the PowerShell. This data-structure limits the commands and the parameters that can be invoked by the user.

Advanced Topics Workshop at LISA14

November 11, 2014, Seattle, WA

Summarized by Josh Simon

Tuesday's sessions included the 20th (and final) Advanced Topics Workshop; once again, Adam Moskowitz was our host, moderator, and referee. Unlike past years, we only ran for a half day. With only two new participants (both longtime LISA attendees), Adam covered each participant's interface to the moderation software in brief one-on-one sessions over lunch. We started with our usual administrative announcements. We mostly skipped introductions. However, Adam noted that two people here were at the first ATW, and he and I were both here for the past 18 years (he as moderator and I as scribe). In representation, businesses (including consultants) outnumbered universities by about two to one (about the same as last year); over the course of the day, the room included 10 LISA program chairs (past, present, and announced future, up from five last year) and nine past or present members of the LOPSA or USENIX boards.

Our first topic, which took two-thirds of the discussion time, was on why this was the last ATW. To oversimplify:

- The workshop has met its originally stated goals of increasing the number of more senior people who attend the conference and to have a space to discuss possibly confidential issues in a non-public venue with other senior people and without interruption.
- Most of the topics we discuss are not controversial and don't lead to much discussion, spirited or otherwise. There were few if any strong opinions.
- Many of the topics were repeated year after year but nothing new was being said.

Of course, since this decision was announced without input from the participants, it generated a very spirited and passionate discussion (and at times an outright debate). That discussion wandered through what the workshop should be if it were to continue, as well as the future direction of the LISA conference itself. No definitive conclusions were reached, in large part because not all stakeholders were present or represented.

It was argued that the workshop has been successful. The founder, John Schimmel, had originally looked at the conference and identified a problem: the more senior system administrators would only come to LISA (which was then more about training junior administrators) if they were speaking or teaching, and were much less likely to come as attendees. The workshop was an attempted solution to that problem: get the more senior sysadmins present for the workshop, where they could have non-public discussions without having to step down the language for junior sysadmins to understand, and they'd be (and were) much more likely to stick around for the rest of the conference.

It was also argued that there's still value in getting together, even if just "at the bar." Many were quick to point out that it would be much more difficult to sell "I'm meeting with a couple of dozen senior sysadmins at the bar" than "... at the workshop" to their management.

Some of the other points we considered during the discussion included:

- What problems are there with the conference today, and how can we solve them? In general, we see a need and provide service to the community; someone therefore needs to present a proposal (for a workshop, talk, tutorial, or whatever is appropriate) and see whether it's accepted.
- If there were no ATW, would you still attend LISA? If not, why not? How can LISA evolve to change your mind? For many people—nearly half of those present—ATW was *the* reason they attend LISA, in part because we provide seniority without overspecialization. Also, would the conference be losing something important? (Most thought so.)
- Is the workshop name still appropriate? Some years the topics aren't necessarily advanced, but we're talking about implementing and integrating existing technologies into existing (often complex legacy) environments.
- A side discussion came up as to whether we're elitist. How many of us sent in a position paper? (Virtually all at least once; it's only required the first time. At least one participant submits a position paper every year.) We need some kind of bar to keep the limited space available for the more senior

people; any bar, even if it's "senior vs. junior," can be perceived as elitist. Perhaps owning up to the word as a function of the workshop would be a good thing. Some present parsed the message of "the workshop is perceived as elitist" as "USENIX would prefer you weren't here"; the USENIX representatives present disagreed with that parsing.

- How do we, as senior sysadmins and often longtime LISA attendees, contribute to the conference?
- Why should USENIX fund this (indeed, any) workshop? Is it an appropriate use of USENIX's time and money? By reserving the room for the workshop, which loses money, there's one less room available for tutorials, which make money. That led to a discussion about what LISA should be and what USENIX should do. It was noted that USENIX has to continue to make money, to serve the community, and to serve the conference. If it's not going to be making money but serving another purpose that serves the community, that's still valid. Nobody at USENIX wants the workshop participants to stop coming, but by pulling the plug they may have that effect. The board members present agreed that they would welcome discussions with a committee elected to consider the future of the workshop.
- We need something formal on the schedule to justify our attendance and travel to our management.
- Some of our extended discussions over the years have spawned their own workshops or conferences (notably the Configuration Management workshop held at LISA and the standalone LISA-NT conference).
- This workshop is a microcosm of what happened to the general conference: it spun off other workshops and conferences and made the general conference look less important and less relevant.
- This forum of knowledgeable and experienced people is a hard resource to replace. Is this workshop, as currently constituted, the right forum for such discussion? If not, what is?
- Is the issue with the format or the lack of new blood? We only get two to three new people requesting to join each year because the position paper scares many off. That said, many agree we need new blood. One wants everyone to encourage someone new to join us next year, doubling the size of the workshop; unfortunately, workshops need to be size-limited for things to work.
- Having the same people year after year can lead to an echo chamber. Something not impacting us may be overlooked or ignored. Some of us show up out of habit; this won't be a problem as long as it doesn't drive away people with specific topics to discuss. Perhaps a position paper should be required from every participant every year (instead of only the first year)?
- How do we bring, to the conference or to the workshop if it remains, other qualified people, those on the leading edge of technologies?
- How can we be better at mentoring leaders?

It was stressed that all interesting proposals (for papers, talks, tutorials, and workshops) are both welcome and desired. A proposal saying "After *N* years we have a new version of the ATW called something else" would be considered as long as it indicated how it would be different. The number of workshops is limited by the number of rooms available and by the number of places any one of us can be at one time. It's not just what should serve USENIX or LISA better but what would serve us (the constituents) better.

As a palate cleanser we went with a lightning round: what's your favorite tool? Answers included Aptly, C+11, CSVKit, Chef, Docker, Expensify, Go, Google Docs, Graphana, Graphite, HipChat, JCubed, JIRA, R, Review Board Sensu, Sinatra, Slack, Git and git-annex, logstash, and smartphone-based cameras.

Our next discussion was about platform administrators. With user-level networking and systems becoming one blended platform, are platform admins the new sysadmins? Is this a new tier for provisioning logical load balancers and front and back ends? The conclusion seemed to be that it's still sysadmin, just a specific focus. It's like any other new technology, and may be due to the extension of virtualization into the network world. The "are we specializing?" question comes up often (e.g., storage, network, Windows versus UNIX, and so on), and we're still sysadmins.

One participant strongly disagreed, thinking platform administration is fundamentally different in that for the first time it's now readily straightforward and easy to think of system deployment as a cheap software call or RPC. It's so lightweight in so many ways that it's fundamentally different from early virtualized environments. His business expects to routinely spin up thousands of virtual instances. How much and how fast to spin things up (and down again) is a game changer. The other part of it is that the environments they're using for this are fundamentally confused about everything of value, with APIs calling APIs. At some level this is sysadmin on a new layer, because it's a programmability block mode; much of the sysadmin stuff is hidden. What happens when you're repairing a cluster and something says you have to scale out from 200 to 1000? Either "you don't" or "you wait" might be the answer.

Another person noted that we're system administrators, not just focused on the single computer (or network or person), but on the interaction between those systems (computers, networks, people, and so on). Nothing's really changed: we still look at the pieces, the goals, and whether the product/service is being delivered as expected.

Two side discussions came out of this as well. First, with virtualization and cloud and *aaS, how many businesses still administer their IT as their core function? Second, sysadmins who won't write code (including shell scripts) will soon be out of a job, since the field is moving towards that: systems will be built by writing code. With virtualization and APIs, we suspect that most sysadmins will fall into the "services" mode, maintaining

services on perhaps-dedicated, probably virtual machines, as opposed to the folks administering the underlying hardware on which the virtualized machines run.

Our next discussion was started with the phrase, "If I had a dollar for every time someone said DevOps was the future..." It took forever for Agile to get into Gartner, but DevOps is there already and, in the speaker's opinion, has jumped the shark in less than two years. DevOps is a horribly abused term, despite being a paradigm shift. At ChefConf, the belief was that DevOps was "software engineers throwing off the yoke of the evil sysadmins who have oppressed them for so long." (That's a direct quote from their keynote speaker.) Code needs to be in the realm of infrastructure; what we did 20 years ago won't scale today. There's a huge difference between writing actual code and writing a Ruby file that consists entirely of declarations.

In another company, they have some developers who do sysadmin work as well, but not all developers there have the background, and the speaker doesn't trust them to do it: their sysadmins are developers but not all developers are sysadmins.

One participant who has been going to DevOps and infrastructure-as-code meetups for a while now says it's like SAGE-AU and Sun Users' Group repeating the same mistakes all over again.

Even now, everyone has a different definition of DevOps, though most could agree it's not a tool, position, mechanism, or process, but a culture, about having the operations folks and engineers talk to each other as the product is written as well as after operations has it in production. There's a feedback loop through the entire life cycle. But having "a DevOps team" is not true; it's about not isolating teams.

We had a brief conversation on recruiting. How do you find and entice qualified people to jump ship to a new company? They have problems finding candidates who want to come to the company. The only response was that sometimes you simply can't, and one participant noted he turned down a great job because of its location (being sufficiently unpleasant to make it a deal breaker).

We then discussed what tools people are using to implement things within a cloud infrastructure. One participant is all in AWS, for example. Do you do it manually or through automation, what do you use to track things and manage things, and so on? One participant snarked he'd have an answer next year.

Another is about to start moving away from the AWS API to the Terraform library (written in Go), which supports several different cloud vendors and has a modular plugin system. Beyond that it depends on what you're trying to do.

Yet another says part of this is unanswerable because it depends on the specific environment. His environment is in the middle of trying to deploy OpenStack storage stuff, and most of the tools can't work because they reflect the architectural confusion thereof. They have used ZeroMQ for monitoring and control due to scalability to a million servers—which is what they call a medium-sized application. Precious few libraries can handle that level. That's the number thrown around by HPC too.

Once you care about speed and latency and measurements you can make a better judgment of how much to spin up to handle those requirements and whether physical or virtual is the right answer for your environment.

Our final discussion topic was on getting useful information from monitoring data.

One participant loves Graphite. Since he has a new hammer everything looks like a thumb, so he's been trying to get more and more into it, and now that he's taken the stats classes, he needs more low-level information so he can draw correlations and eventually move data out of the system. What are others doing with their statistics? What are you using to gather, store, and analyze data? In general, R and Hadoop are good places to start, and there's an open source project called Imhotep for large-scale analytics. Several others noted they use Graphite as a front end to look at the data. Spark is useful for real time and streaming. Nanocubes can do real-time manipulation of the visualization of a billion-point data set. Messaging buses discussed included RabbitMQ and ZeroMQ.

How does this help? In one environment, they used the collected metrics to move a datacenter from Seattle to San Jose, and the 95th percentile improved a lot. Another noted that Apple determined that the transceiver brand makes a huge difference in performance.

We wrapped up with the traditional lightning round asking what we'd be doing in the next year. Answers included an HPC system with 750K cores and an 80 PB file system, automation and instrumentation, chainsaws and hunting rifles in Alaska, enabling one's staff, encouraging people to create and follow processes, exabyte storage, functional programming, Hadoop, home automation, Impala, infrastructure, learning a musical instrument, merging an HPC-focused staff into the common IT group, moving from GPFS to something bigger, network neutrality, organizing a street festival and writing the mobile app for it, packaging and automated builds, producing a common environment across any type of endpoint device, R, scaling product and infrastructure (quadrupling staff), Spark, trying to get the company to focus on managing problems not incidents, and updating the Cloud Operational Maturity Assessment.

Our moderator thanked the participants, past and present, for being the longest-running beta test group for the moderation software. The participants thanked Adam for moderating ATW for the past 18 years.

29th Large Installation System Administration Conference (LISA15)

Sponsored by USENIX, the Advanced Computing Systems Association, in cooperation with LOPSA

November 8–13, 2015, Washington, D.C.

Important Dates

- Submissions due: April 17, 2015, 11:59 pm PDT
- Author response period: May 11, 2015–May 17, 2015
- Notification to authors: June 5, 2015
- Final papers and poster abstracts due: August 27, 2015, 11:59 pm PDT

Conference Organizers

Program Co-Chairs

Cory Lueninghoener, Los Alamos National Laboratory

Amy Rich, Mozilla Corporation

Overview

Publishing at USENIX LISA Means Industry Impact and Recognition

Papers published at LISA receive visibility and recognition from a unique audience that fundamentally builds and operates the systems that run the world. For a researcher, publishing at LISA proves that your work is relevant and applicable to industry and has the potential to transform the profession's approach to systems engineering.

USENIX Stands for Open Access and Advancing the State of the Computing World

USENIX strongly believes that research is meant for the community as a whole and maintains a clear stance advocating open access. Your poster abstract or paper and presentation will be available online at no charge, where it can have the most impact and reach the broadest possible audience. Papers and poster abstracts will also be part of the conference proceedings, which has its own ISBN.

Topics of Interest

- Systems and Network Engineering
  - Cloud and hybrid cloud computing
  - Software Defined Networks (SDN)
  - Virtualization
  - HA and HPC clustering
  - Cost effective, scalable storage
  - Configuration management
  - Security
- Monitoring and Metrics
  - Performance and scalability
  - Monitoring, alerting, and logging systems
  - Analytics, interpretation, and application of system data
  - Visualization of system data
- SRE/Software Engineering
  - Software-defined System Administration
  - Continuous delivery and product management
  - Continuous deployment and fault resilience
  - Release engineering
  - Big Data
- Culture
  - Business communication and planning
  - Standards and regulatory compliance
  - DevOps
  - On-call challenges
  - Distributed and remote worker challenges
  - Mentorship, education, and training
  - Workplace diversity

Papers

Refereed papers accepted to LISA describe new techniques, tools, theories, and inventions, and present case histories that extend our understanding of system and network administration. They present new ideas, backed by rigorous and repeatable methodology, in the context of previous related work, and can have a broad impact on operations and future research.

Accepted papers will be published in the LISA15 proceedings and conference web pages, and all accepted papers will be given a presentation and Q&A slot in the LISA15 academic research track. After acceptance, paper authors should also create a companion poster to present during the poster session to further engage with the conference attendees. Cash prizes will be awarded at the conference for the best refereed paper.

Posters

The LISA juried posters session is a great opportunity to present your work visually, either as a companion to a refereed paper or as a standalone submission displaying work in progress. Posters present the core of your research to a wide audience, providing you with the ability to gain valuable feedback from both industry and academic audiences. Poster abstracts of 500 words or less will be published in the LISA15 Proceedings, and a link to your research website will be included in the conference web page. Additionally, cash prizes will be awarded at the conference for the best juried poster.

We hope that you will consider sharing your ongoing research with the LISA community. All paper and poster abstract submissions are due by April 17, 2015. If you have any questions beyond this CFP, please contact lisa15research@usenix.org.

Paper and Poster Submission Rules

Submissions may only be submitted by an author, no third parties. Authors retain ownership of the copyright to their works as described in the USENIX Conference Submissions Policy at www.usenix.org/usenix-conference-submissions-policy.

Submissions whose purpose is to promote commercial products or services will be rejected.

Papers and poster abstracts may not be simultaneously submitted to other venues. Writing must be original and not previously published online or otherwise. If the content is similar to, or builds upon prior work, the author must cite that work and describe the differences.

The first page of each paper submission or poster abstract submission must include the name, affiliation, and email address of the author(s).

All paper submissions must be full papers, in draft form, 8–12 pages in length, including diagrams, figures, full references, and appendices. All poster abstracts must be 500 words or less.

All draft papers and poster abstracts must be submitted as PDFs via the Web form, which is located at https://lisa15.usenix.hotcrp.com/. They must be in two-column format, using 10-point type on 12-point (single-spaced) leading, with a maximum text block of 6.5" wide x 9" deep, with .25" inter-column space, formatted for 8.5" x 11" paper. Pages must be numbered, and figures and tables must be legible when printed. Templates are available for Microsoft Word and LaTeX at www.usenix.org/templates-conference-papers. Papers not meeting these criteria will be rejected without review.

Paper and Poster Acceptance Details

All submitters will be notified of acceptance or rejection by June 5, 2015.

Authors of accepted papers will be assigned one or more shepherds who will read and offer advice on intermediate drafts before submission of the final paper. At least one author must present at the LISA conference, and the chosen presenter(s) must rehearse their presentation with their shepherd prior to the conference.

One author per accepted paper will receive a registration discount.

Final poster abstracts and papers must be submitted for publication by August 27, 2015, and completed posters are required by the start of the conference. Accepted presenters should ensure that they have enough time to acquire the necessary approvals through their organizations' Institutional Review Board (IRB) or similar process in time for the final paper submission deadline.

If any accepted presenters need an invitation letter to apply for a visa to attend the conference, please identify yourself as a presenter and include your mailing address in an email to conference@usenix.org as soon as possible. Visa applications can take at least 30 working days to process.

All accepted papers and poster abstracts will be available online to registered attendees before the conference. If your work should not be published prior to the event, please notify production@usenix.org when you submit your final copy.

usenix

LISA15

Nov. 8 – 13, 2015 | Washington, D.C.

Sponsored by USENIX in cooperation with LOPSA

SRE

CULTURE

METRICS

Industry Call for Participation

LISA is the premier conference for IT operations, where systems engineers, operations professionals, and academic researchers share real-world knowledge about designing, building, and maintaining the critical systems of our interconnected world.

We invite industry leaders to propose topics that demonstrate the present and future of IT operations. LISA submissions should inspire and motivate attendees to take actions that will positively impact their business operations.

Important Dates

Submissions due: **April 17, 2015, 11:59 pm PDT**
Notification to participants: **June 5, 2015**

Topic Categories

Systems and Network Engineering

- Cloud and hybrid cloud computing
- Software Defined Networks (SDN)
- Virtualization
- HA and HPC clustering
- Cost effective, scalable storage
- Configuration management
- Security

Monitoring and Metrics

- Performance and scalability
- Monitoring, alerting, and logging systems
- Analytics, interpretation, and application of system data
- Visualization of system data

Conference Organizers

Program Co-Chairs

Cory Lueninghoener, Los Alamos National Laboratory
Amy Rich, Mozilla Corporation

SRE/Software Engineering

- Software-defined System Administration
- Continuous delivery and product management
- Continuous deployment and fault resilience
- Release engineering
- Big Data

Culture

- Business communication and planning
- Standards and regulatory compliance
- DevOps
- On-call challenges
- Distributed and remote worker challenges
- Mentorship, education, and training
- Workplace diversity

Proposals we are Seeking

- **Talks:** 30-minute talks with an additional 10 minutes for Q&A
- **Mini-tutorials:** 90-minute courses teaching practical, immediately applicable skills
- **Tutorials:** Half-day or full-day courses taught by experts in the specific topic, preferably with interactive components such as in-class exercises, breakout sessions, or use of the LISA lab space
- **Panels:** Moderator-led groups of 3–5 experts answering moderator and audience questions on a particular topic
- **Workshops:** Half-day or full-day organizer-guided, round-table discussions which bring members of a community together to talk about common interests
- **Papers and posters:** Showcase for the latest research; see the Academic Research Papers and Posters Call for Participation at www.usenix.org/lisa15/cfp/papers-posters for more information

Submit your proposal today!
www.usenix.org/conference/lisa15/call-for-participation/submission

If You Use Linux, You Should Be Reading *LINUX JOURNAL*™

- In-depth information providing a full 360-degree look at featured topics relating to Linux
- Tools, tips and tricks you will use today as well as relevant information for the future
- Advice and inspiration for getting the most out of your Linux system
- Instructional how-tos will save you time and money

Subscribe now for instant access! For only $29.50 per year—less than $2.50 per issue—you'll have access to *Linux Journal* each month as a PDF, in ePub format, in Mobi format, on-line and through our Android and iOS apps. Wherever you go, *Linux Journal* goes with you.

SUBSCRIBE NOW AT:

www.LinuxJournal.com/subscribe