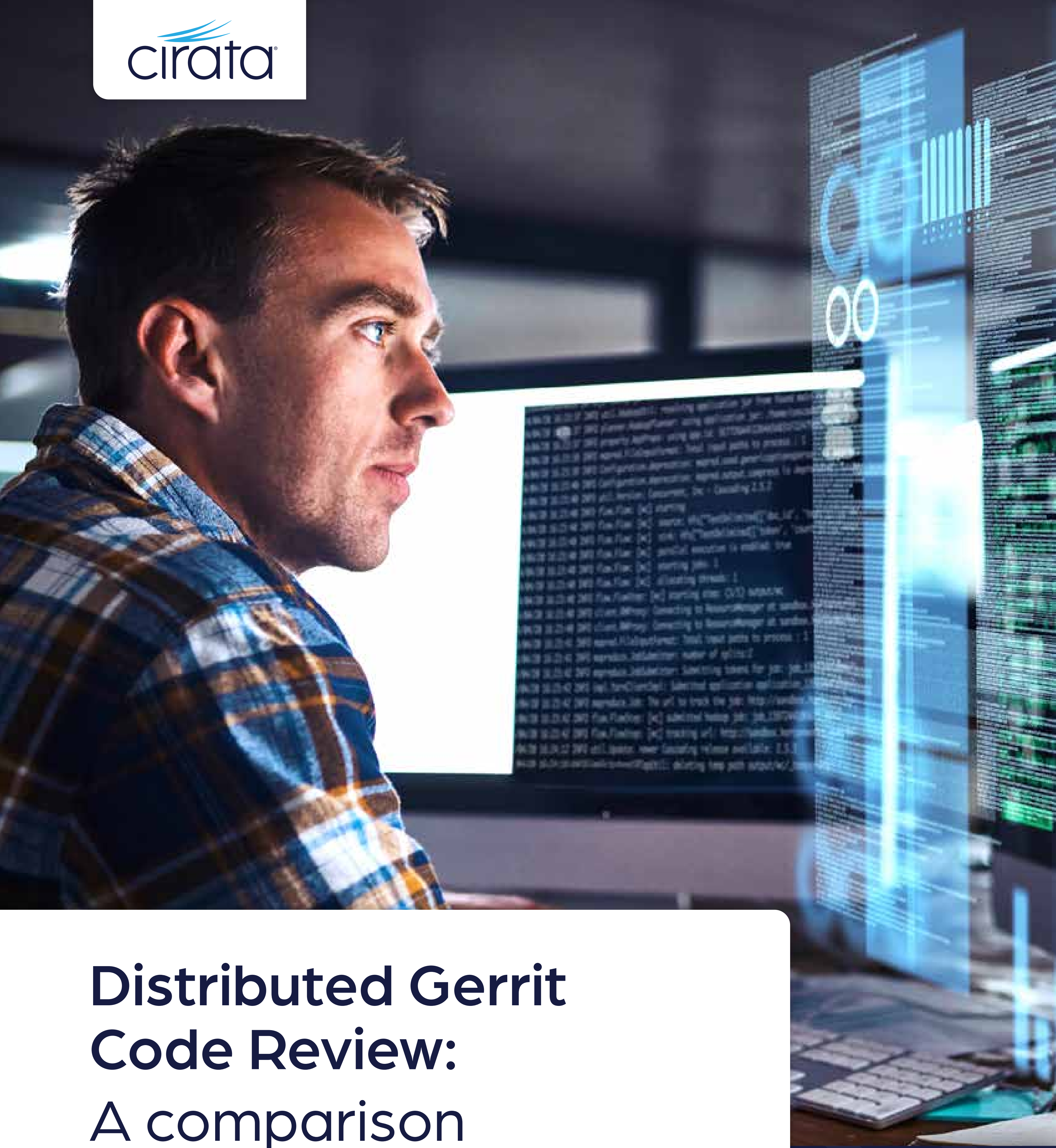


Distributed Gerrit Code Review: A comparison

Technical white paper

Distributed Gerrit Code Review: A comparison

Introduction

Gerrit Code Review, an open-source tool originally developed by Google, streamlines the code review process for software teams using Git. It keeps a detailed record of all code reviews directly within your project's history. This includes valuable insights like comments from your team, automated system checks, and even the evolution of code changes – including those from abandoned reviews. This level of transparency not only helps you understand the 'why' behind your software's development, but also ensures compliance with government regulations that require traceable change control.

Gerrit multiple primary deployments

Traditionally, Gerrit was designed to operate on a single server. While Google has adapted Gerrit to work seamlessly across multiple locations using their own internal technologies, this solution isn't available to the wider community.

Recognizing the need for distributed capabilities, the Gerrit open-source community has been actively seeking solutions.

Cirata stepped in to address this challenge, developing Gerrit MultiSite. By leveraging Git MultiSite and our patented DConE technology, Gerrit MultiSite empowers teams to collaborate effortlessly across multiple locations. It ensures smooth, synchronized replication of everything from Git repositories and Gerrit events to other critical data, no matter where your team members are located.

The comparison

The open-source community has been actively working on the "multi-site" plugin as a way to enable distributed Gerrit capabilities. However, this plugin is only one piece of the puzzle. In reality, you need at least four different plugins to achieve a fully functional, multi-site Gerrit setup with multiple primary instances.

We've provided a detailed ***overview of the architecture***. We'll be referencing this diagram and the implementation details throughout this whitepaper as we compare each component to Cirata's Gerrit MultiSite solution.

The 4 critical optional components are:

1. Multi-site plugin
2. Broker plugin
3. Global Ref-DB plugin
4. Replication plugin (either push or pull)

If your Gerrit deployment wants to have a "single service presence" then 2 additional optional plugins and 1 additional service will be necessary:

5. Health check plugin
6. HA Proxy
7. Web-session broker plugin

“multi-site” component vs. codified changes to Gerrit

Back in the early days of Gerrit version 2, we explored using plugins to achieve the level of replication needed for a truly enterprise-ready solution. However, the existing plugin capabilities fell short. To meet the demands of our customers, we decided to create our own customized version of Gerrit, focusing on essential enhancements.

We keep a close eye on Gerrit's ongoing development, always looking for opportunities to integrate our solution into a plugin as opposed to a full rebuild of Gerrit. While the gap is narrowing, our implementation still requires certain features not yet available in the standard Gerrit distribution. As such, we continue to maintain our own optimized version to deliver the best possible experience.

JGit modifications

Content delivery policy

Cirata enhances Gerrit by modifying JGit to intercept repository updates. We ensure these changes are instantly replicated and kept in sync across all your locations using Git MultiSite (our improvements to JGit are openly shared on GitHub after each release).

By intercepting repository updates, we deliver:

1. Data for the update can be packaged up for parallel delivery to other systems.
2. Data for the update can be required to be delivered to at least N other systems (tunable, defaults to 1).

The content delivery policy in Step 2 is crucial for preventing disruptions in your distributed Gerrit environment. In a distributed ecosystem, it's possible for the system to initiate an update even if the required data resides only on a single server. If that server becomes unavailable before the other sites receive the data, the entire update process across all locations comes to a standstill. The Content Delivery Policy safeguards against this by ensuring data is properly distributed before any updates proceed, minimizing the risk of single points of failure and keeping your Gerrit system resilient.

Once the Content Delivery Policy is met, ensuring data is available across your system, updates are then proposed and voted on using a consensus mechanism (Paxos). Each approved update is then scheduled for delivery to the appropriate replica family. This schedule is simply a numbered sequence, the “Global Sequence Number” (or GSN), unique to each replica family. Each replica family has its own set of members participating in the voting process and maintaining its own independent sequence.

Idempotent update operations

When recovering from an application failure, it's sometimes necessary to re-run an update operation because it's not always clear if it was completed successfully the first time. This means that an update, identified by its GSN, might need to be executed multiple times, always in order and without any other operations in between.

Let's illustrate this with a simple example: imagine you want to update a value stored on a server from 15 to 20.

- **The non-idempotent way:** The update instruction is to “add 5 to the value.” The first time, you get the correct result of 20. But if you run it again, the value becomes 25, then 30, and so on, leading to incorrect data.
- **The idempotent way:** The update instruction should be to “set value from 15 to 20”. No matter how many times you run it, it will only update from the old value 15 to the new value 20 once. This ensures that you do not change an incorrect value of 13 to a new value 20 as part of this operation.

When dealing with repository operations in a distributed system like Gerrit, it's critical that these update operations are designed to be idempotent, just like in the example. This ensures data integrity and consistency, even in the face of unexpected interruptions.

To ensure smooth recovery from any system interruptions, we made sure that all JGit repository update operations could be safely repeated if necessary. Most of these operations were already idempotent. However, we identified that “atomic batch updates” needed some adjustments to achieve the same level of reliability. We implemented the necessary changes to ensure these updates could also be replayed without causing any issues.

Broker libModule and plugins

The open-source Gerrit multi-site solution uses a specific plugin, called the ***broker libModule*** along with associated events plugins (see below) to deliver Gerrit updates like index events, cache events, and stream events across different sites. This plugin operates independently from the mechanism used to synchronize repository changes (like Git push replication).

This separation can lead to many race conditions where updates might arrive out of order. For instance, information about changes to the Gerrit index might reach a site before the actual repository data it references. To address this, each plugin implementation needs to include complex mechanisms to retry events and ensure they're processed in the correct sequence. This often involves using a queue where updates are insert-sorted and validated before being applied.

Cirata streamlines the process by using the same efficient mechanism for both repository updates and Gerrit events. This ensures that any changes to the repository itself are always delivered before any related events, eliminating the risk of timing conflicts. This simplified approach results in a much more straightforward and reliable queue processing system.

We pay special attention to events that involve multiple repositories, although these are far less common than events within a single repository. To handle these specific cases, we've built a backoff/validate/execute mechanism that carefully checks and processes these events, ensuring they're executed correctly even if they arrive slightly out of order.

As Gerrit continues to evolve, we're seeing fewer and fewer of these cross-repository events. We anticipate that soon they'll be eliminated entirely, allowing us to further simplify our solution.

Currently, there are 5 open source Gerrit broker plugins that are offered to fulfill this service requirement:

- ***events-aws-kinesis***
- ***events-eiffel***
- ***events-gcloud-pubsub***
- ***events-kafka***
- ***events-rabbitmq*** (not built by default)

Gerrit Broker plugin class limitations

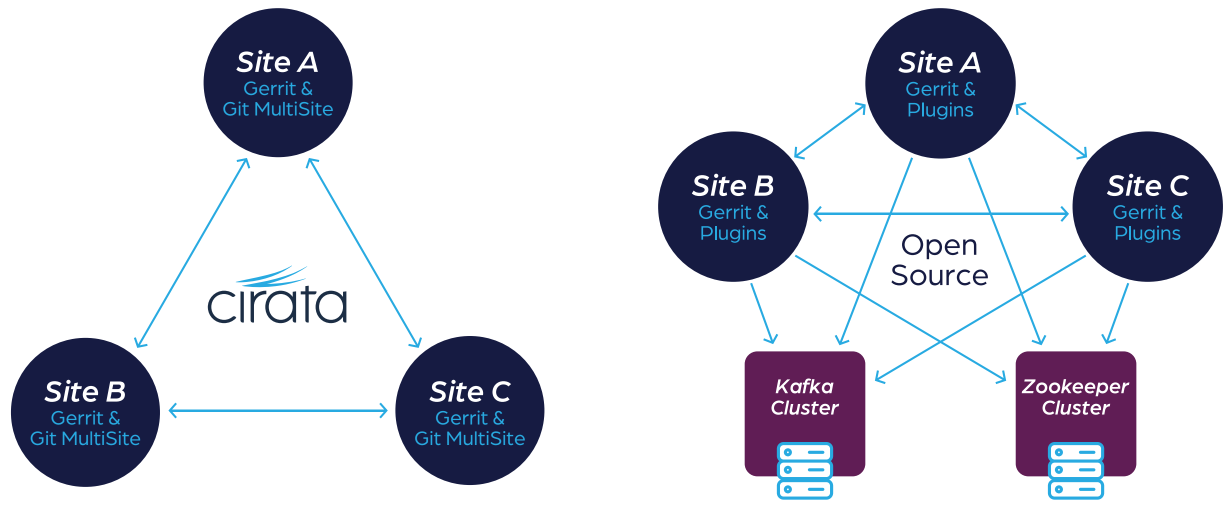
The five plugins operate on a "publish/subscribe" (or "pub/sub") model. This model introduces a challenge because each plugin establishes its own network connection, separate from the Gerrit servers themselves. Most of these plugins necessitate a centralized service. Although this service can be made highly available (for instance, through a Kafka cluster), its central nature creates a vulnerability: reliance on network connectivity. If the data center hosting the centralized service becomes isolated from your remote sites' networks, the service becomes inaccessible at those locations, halting updates until connectivity is restored.

Cirata's Gerrit MultiSite leverages the full capabilities of our Paxos-enabled Git MultiSite product, deployed alongside each Gerrit instance. This eliminates the need for additional plugins and, crucially, avoids introducing any extra network dependencies. Replication across subsets of servers (known as "replication groups" or "RGs") remains consistent as long as a voting majority is maintained. Should a subset of members lose their voting majority due to a network outage, they will temporarily transition to a read-only state until connectivity is restored. Any unavailable members will automatically synchronize their data upon reconnection.

Comparison of 3 site server requirements

Let's imagine a Gerrit deployment across three sites globally. Cirata's Gerrit MultiSite solution necessitates just one server at each site. As long as any two of these sites maintain network connectivity, they remain fully operational. If the third site is also connected, it too functions seamlessly. However, if the third site experiences a network outage, it will temporarily switch to a read-only mode until connectivity is restored.

The following diagram shows the resources necessary for the Cirata MultiSite solution (left) versus the open source "multi-site" solution (right) for a replicated three Gerrit server deployment:



Now, let's examine the same three-site setup using only open-source Gerrit plugins. While you'll still need the same three Gerrit servers, one at each site, this approach also requires additional infrastructure: a Kafka cluster and a Zookeeper cluster, both set up for redundancy. (Refer to the diagram above for a visual representation). Kafka and Zookeeper are depicted as "Kafka Cluster" and "Zookeeper Cluster" because they can't be distributed across a wide area network; they need to reside within a data center. Although they could technically be located at any of the original three sites, it's crucial to remember that any Gerrit server losing connection to either of these services will become partially or fully inoperable. This setup inherently introduces more potential points of failure compared to the Cirata solution.

Operational Details

Returning to the limitations of the Gerrit Broker Plugins, there's a noticeable lack of guidance on how to effectively deploy and manage them in a real-world, distributed environment. Critical operational details, particularly around handling network outages, resource constraints, server failures, server availability, scaling the system, or adding new servers, are scarce. With five different plugin types involved, acquiring the expertise to troubleshoot and maintain this setup can be a complex and costly endeavor.

Opting for cloud provider services like AWS or Google Cloud can simplify the management overhead associated with a self-hosted solution. However, it's important to consider the potential trade-off of vendor lock-in, where you become reliant on a specific cloud vendor's infrastructure and services.

Global RefDB plugin

The Global RefDB plugin is a critical component of the open-source multi-site solution for managing repository updates. It prevents ***split-brain*** scenarios, where conflicting changes occur, by ensuring that only one of the multiple primary servers can update a Git reference from the "current" hash value to a "new" hash value via an "atomic test and set" operation.

If a primary server attempts an update but is unsuccessful:

- The client attempting the update will receive a notification of the failure
- The server needs to fetch the latest data from the primary server that successfully completed the update
- The client will need to refresh their local copy of the repository before attempting the update again.

A key challenge with this open-source approach is that a single point of failure exists at a critical point during repository updates. If the Global RefDB is updated, but the server holding the corresponding data becomes inaccessible (due to a crash, network issue, etc.), other primary servers cannot obtain the necessary Git repository data to continue their operations. This creates a vulnerability that Cirata's Content Delivery Policy specifically addresses.

We've encountered situations where this kind of scenario severely disrupts workflows. Cirata's solution proactively prevents this by ensuring data is replicated across multiple servers before any updates are scheduled. This eliminates the risk of a single server outage halting your entire system.

If such a failure were to happen without this safeguard, restoring the system to a consistent state becomes a complex operational task, particularly if the server remains permanently unavailable. It would necessitate checking all Global RefDB entries and potentially resetting them to a previous state, which can be time-consuming and error-prone. In some cases it may even result in data loss in the case of an irrecoverable server.

Another potential issue arises if the Global RefDB table is updated, but for some reason, the corresponding change isn't applied to the actual repository. This could happen if the file system becomes read-only due to a hardware problem, or if the network connection to the RefDB service is lost, preventing the plugin from receiving confirmation of the update. Without making the RefDB update idempotent (i.e., ensuring it can be safely repeated without adverse effects), the only way to resolve this mismatch is to revert the RefDB back to its previous state. (Note: As of the time this was written, the doCompareAndPut() operation in the open-source plugin isn't coded to be idempotent.)

Resolving such inconsistencies between the repository's own RefDB and the Global RefDB table would necessitate specialized administrative tools, which we haven't been able to find any reference to online.

Global RefDB plugin class limitations

Similar to the Broker Plugins, this entire category of plugins faces the same networking challenges we discussed earlier. Each plugin necessitates setting up, maintaining, and potentially paying for a separate service (e.g., on platforms like AWS or Google Cloud). Once again, comprehensive operational documentation for these plugins is scarce, if not entirely absent. (Refer to the "Operational Details" section above for more information on this challenge).

Specific plugin implementation limitations

Given the critical nature of the Global RefDB Plugin we will look at each in turn to discuss, briefly, some of their limitations.

RefDB Plugin: aws-dynomodb-refdb

Using AWS DynamoDB as the underlying database for this plugin might be problematic due to its "eventual consistency" model. While this term suggests a slight delay in data updates, DynamoDB's behavior can lead to data being overwritten in order to achieve consistency across its distributed system. In the context of the RefDB, overwriting a key-value pair could have severe consequences, unlike in applications like shopping carts where an incorrect update might just result in an incomplete purchase.

It's crucial to avoid configuring a multi-region DynamoDB Global Table when using this plugin. Doing so could lead to ***data loss***, which, in this context, translates to repositories becoming inconsistent or "split-brain" – a highly undesirable situation where different servers have conflicting versions of the same repository.

However, the limitation of not being able to utilize DynamoDB Global Tables means that you're restricted to a single region for your database. This can pose challenges in a global deployment, potentially leading to network connectivity issues and increased latency for users located far from the database region.

RefDB plugin: spanner-refdb

This plugin isn't included in the standard Gerrit distribution, so you'll need to compile it yourself. While the Google Spanner implementation might offer comparable performance to Cirata's Paxos solution, it comes with certain considerations. You'll either need to host your Gerrit servers on Google Cloud to ensure optimal connectivity, or accept the potential for network-related issues if your Gerrit servers are located elsewhere. Additionally, keep in mind that using Spanner involves utilizing Google Cloud services and incurring associated costs.

RefDB plugin: zookeeper-refdb

The Zookeeper implementation is not hardened for WAN networking issues. That means it will need to be a centralized service with all of the networking connectivity issues already discussed above.

Repository updates

In open-source Gerrit, keeping repositories synchronized between primary sites is handled through either "pull" or "push" plugins. Setting up this synchronization can be intricate, not just because of the many configuration options for timeouts, retries, and other settings. You also need to carefully choose which repositories and even specific branches and tags (called "refs") within those repositories get replicated. This level of control is necessary to accommodate a wide range of complex scenarios, but it adds to the overall setup complexity.

Selective replication

Cirata Git MultiSite guarantees that every repository under its management has all its branches and tags replicated across all locations. This ensures each replica is an exact replica, eliminating any inconsistencies. There's no need to configure any special rules to replicate important data like change refs or NoteDB information. Additionally, the concept of Replication Groups (RGs) makes it simple to set up selective replication, where only specific repositories are synchronized between certain locations. Symmetric replication, where all sites have identical data, is even easier to configure as it involves just a single Replication Group.

Plugin: replication (push replication)

The ***replication plugin*** was the first implemented. Performance issues triggered the development of the pull replication plugin.

Plugin: pull replication

The ***pull-replication*** plugin was implemented to speed up replication and from reports it has accomplished this objective.

The current recommended approach for achieving multiple primary replications in open-source Gerrit involves using the pull-replication plugin. This plugin is triggered under one of three specific conditions:

1. When notified by another Gerrit server that the repository has been updated (an HTTP POST)
2. Whenever a primary determines that it is out-of-date with respect to a ref via a failed Global RefDB attempted update
3. When Gerrit is started (configurable) or when the plugin is reloaded (configurable)

For the first condition (#1) to work, each primary server needs to be explicitly configured with a list of its "neighbor" servers that should be notified whenever there's a repository update. This server-by-server configuration can be time-consuming and prone to errors. Mistakes in this setup can lead to disruptions, requiring manual intervention by administrators to get things back on track for your users.

Currently, achieving robust and reliable replication ***requires configuring up to three dozen settings***. These settings can be quite intricate, including specifying credentials (like SSH or HTTP) for each remote server. Since wide area network (WAN) traffic behaves differently than local area network (LAN) traffic, some configurations, such as timeouts, might need to be fine-tuned through trial and error. As mentioned earlier, readily available operational guidance for these settings is limited.

Let's look at a couple of examples:

gerrit.replicateOnStartup

Although typically false, enabling this setting can impact scalability. The system's performance may be affected by the combination of the number of repositories you're replicating and the number of other primary servers involved in the process.

replication.lockErrorMaxRetries

According to the documentation, if several primary servers simultaneously attempt to pull an update from this primary server, locking mechanisms might prevent one or more of them from succeeding. This can create a scalability bottleneck as the number of primary servers increases, potentially requiring adjustments to retry settings to ensure updates are eventually successful.

Cirata's Gerrit MultiSite, as explained earlier, leverages Git MultiSite to immediately replicate every repository update to all family replicas. Repositories that are not modified generate zero replication overhead or network traffic, even during system startup. Our servers establish secure connections with each other and utilize a specialized set of protocols for efficient replication management.

Importantly, no manual configuration is needed for replication. When you add new repositories, they're typically assigned to the appropriate replication group automatically based on their names. This significantly streamlines the process of selective replication, where you only want certain repositories synchronized between specific locations.

Additional services

As we mentioned earlier, achieving a "single service presence" requires configuring two additional plugins and one more service. This is a significant advantage for companies offering Gerrit as a service. It allows them to provide a single web address (FQDN) to their users globally. The system then intelligently directs each user to the Gerrit instance closest to them, ensuring optimal performance and responsiveness regardless of their location.

Health check plugin

The health check plugin is essential to the single service presence since the HA Proxy, GSLB, AWS Route 53 or other global load balancing service must know whether or not the Gerrit service is "up" at a site before directing the client to that site.

HA Proxy

Although the open-source Gerrit architecture recommends HAProxy, it's worth noting that other options exist for global load balancing. You could utilize an on-premises GSLB (Global Server Load Balancing) solution, AWS Route 53, or similar services. However, it's crucial to configure these services carefully to ensure that a user is consistently directed to the same Gerrit server, unless that server becomes unavailable.

The standard "round-robin" configuration, which distributes requests evenly across servers, isn't suitable for distributed Gerrit setups. In such environments, some primary servers might be slightly behind others in terms of data updates. A pure round-robin approach could lead to users receiving inconsistent or outdated information. This applies equally to both Cirata Gerrit MultiSite and the open-source Gerrit solution.

Web-session broker plugin

The web-session broker plugin replicates web session tokens to allow already authenticated web browser client sessions to be honored at different primary sites. If a set of primary servers are located within a small region from a set of clients (low latency) and are automatically selected by an "HA Proxy"-like service then this type of service makes some sense.

However, in a globally distributed 'multi-primary' setup, switching users between servers located in different parts of the world can lead to noticeable delays and a less seamless experience. In such cases, requiring re-authentication might be a better indicator of a potential issue rather than a strict necessity.

Currently, Cirata Gerrit MultiSite doesn't replicate web-session tokens. We're actively gathering feedback from our customers to determine if this feature would be beneficial in future versions of our solution.

Cirata Plugin Forks

Git LFS Plugin

The open-source Git LFS plugin enables the location for Git LFS objects to be either Amazon S3 or "local to the server". And the plugin is not "multiple primary hot". That means that you either need to pay Amazon to use their AWS S3 storage or only use a single server (or a set of local servers using NFS to mount the LFS storage area).

Cirata has forked the Gerrit Git LFS plugin in order to enable full replication of the Git LFS data for the "local to the server" case. The LFS data only goes to the same sites as the repository that uses that data. While this will slightly delay a push that requires uploading of Git LFS data based on the Content Delivery Policy, it means that any subsequent checkouts of a replicated repository worldwide will be able to fetch the LFS data from the "local" server – a massive win for globally distributed teams and also for DevOps automation.

Delete Project Plugin

Cirata has forked the Gerrit delete-project plugin in order to require archiving of repositories before they are deleted. This enables a mistaken deletion to be easily recovered including every last change before the deletion occurred. The process of archiving is coordinated by Git MultiSite.

Git MultiSite Administration

While the Cirata Git MultiSite co-product provides the replication services for Gerrit MultiSite, it also provides additional site and global administration benefits via the Git MultiSite UI. There is no equivalent set of features in the open-source Gerrit multi-site software.

- See the replication status of any server via a dashboard.
- See the replication status of specific replication groups.
- See the replication status of a repository.
- Request a "consistency check" for a repository.
- See all the repositories that are replicated to that server (or a subset via filtering).
- Configure events to send notifications to administrators.
- REST API available for automation.
- Administrator's user guide is available.
    - Site addition and removal operations are documented.
    - Repair operations are documented.

Repository consistency checking

In the list above we mention the "consistency check" operation. This operation gathers sufficient information about each replica in the replica family and compares that information. The resulting summary will either prove that the replicas are all consistent - or not. If not, the repository family will go "read-only" in order to enable a Git MultiSite administrator to repair the divergence via the procedure documented in the Administrative User Guide. Divergence is not due to split-brain - which cannot happen due to the Cirata architecture - but normally due to some negative hardware event. Note: The consistency check operation is a "distributed operation" that uses "distributed agreements" to obtain the consistency information at the same GSN and enable direct comparison of the resulting data.

Summary

The above comparison shows the difference between Cirata's enterprise-ready Gerrit MultiSite product and the open-source Gerrit open source parallel effort.

Key benefits of the Cirata MultiSite product:

- No single point of failure.
- Full primary-primary hardened WAN implementation.
- Easy to configure either Symmetric or Selective replication.
- Git LFS "globally local" implementation.
- Git MultiSite administration capabilities.
- Greatly reduced deployment complexity.

Glossary

Acceptor

This is a Paxos term that identifies a node that can vote on a proposal.

Consistency checking

In a distributed service, Consistency Checking validates that a "replicated object" is consistent at all sites. In Git MultiSite Git repositories can be checked for consistency.

DConE

Cirata's patented implementation of Paxos.

Eventual consistency

In its original form, this was used to describe algorithms that will resolve issues between distributed database entries over time. In this case "consistency" can trigger database entry removals and therefore other downstream effects. A simple example is a dynamic multiple-site order entry system that is supposed to know how many items it can sell. If the system sells too many copies then it can trigger some shopping carts to be emptied of the items or, after the item is purchased put it on backorder or cancel the order. For more information, see the Wikipedia article ***here***.

The Gerrit multi-site architecture documentation uses ***eventual consistency*** with a different definition. Instead, what they appear to mean is, over time, any missing data from a Git repository "replica" will be filled in from another "replica" via either a pull or push event. Clearly, "losing" Git information is not what they are talking about so we will not be using the terms "eventually consistent" or "eventual consistency" here. See "Replica Family" below for a better understanding.

FQDN

A Fully Qualified Domain Name is a human-readable name that must be resolved by the Domain Name Service (DNS) to obtain the network address for a server or service.

Global Sequence

The "Global Sequence" is a series of integers. Each integer, called a GSN, represents an idempotent update to a distributed object maintained by a single DSM. In Git MultiSite each repository has its own DSM and therefore its own Global Sequence.

GSN or Global Sequence Number

See "Global Sequence" above.

Idempotence / Idempotent

A property of an operation such that performing the operation multiple times in a row will yield the same result as if it had been executed only once. This includes not only the state of the system that the operation is acting upon but also any and all returned values.

Learner

This is a Paxos term that identifies any node that wants to learn the outcome of an Agreement step.

Membership

A term that is used to describe sets of nodes that are connected to each other as part of a replication group (or other DSM)

Multiple primary

In a distributed application this means that each instance of the application can request changes to the underlying data. The details and characteristics of the replication of those changes depend on the implementation mechanism. Multiple primary is the gold standard.

Paxos

A family of protocols for obtaining consensus between service entities in a distributed network.

Proposer

This is a Paxos term that identifies a node which can propose changes. As well as proposing changes the proposer is also involved in conflict resolution as part of the voting process. Resolution comes from interactions between proposers and acceptors and the losers are expected to re-propose.

Replica

A repository instance that is maintained by Git MultiSite to be "single copy identical" with the other instances of the same repository when their GSN's are identical.

Replica family

The complete set of repository instances that are maintained by Git MultiSite via a single DSM to be "single copy identical" when their GSN's are identical.

Selective replication

Different repository families are resident on different subsets of the total set of servers. Compare with Symmetric replication below.

Single service presence

A single service FQDN that can be used to access a service regardless of where the actual servers for that service is located. Instead of the FQDN representing a specific server, it represents a service and the resolution of the FQDN will select one among the servers providing that service for the client to connect to. Ideally the server selected is the smallest latency from the client that is "up" at the time of the request.

Symmetric replication

All repositories are replicated to all servers. Compare with Selective Replication above.

About Cirata

Leveraging our patented technologies, including the Distributed Coordination Engine ("DConE®"), and trusted by global brands and industry leaders for more than 15 years, our DevOps solutions integrate effortlessly with your existing source code management to increase security, minimize risk, reduce latency, and improve collaboration across globally distributed development teams. In addition, Cirata specializes in the migration of Hadoop data lakes into leading cloud platforms to enable game-changing Artificial Intelligence ("AI") and analytics. With Cirata, data leaders can leverage the power of AI and analytics across their entire enterprise data estate to freely choose analytics technologies, avoid vendor, platform, or cloud lock-in while making AI and analytics faster, cheaper, and more flexible. Cirata's portfolio of products and technology solutions make strategic adoption of modern data analytics efficient, automated, and risk-free.

For more information on Cirata, visit www.cirata.com.