nimbella

A technical whitepaper on Nimbella Serverless Platform

Nimbella

www.nimbella.com

Contact: info@nimbella.com

Table of Contents

Introduction

The Nimbella Serverless Platform is designed to address the challenges developers face when adopting a serverless computing model. There is little doubt that serverless is radically changing the way cloud applications are developed. For the enterprise developers that have already taken notice, they've realized dramatic gains in agility, focus, and costs. Using a serverless architecture, smaller teams can deliver large-scale distributed applications faster and more securely than ever before.

Serverless applications are ones that don't require developers to provision or manage servers or their associated software and hardware infrastructure. Developers are relieved of any responsibility for the operating system, access control, provisioning, right-sizing, scaling, and availability of the infrastructure. Rather, developers are free to focus on core product and business logic.

Nimbella provides all developers (front-end, back-end or full-stack) with a unified programming experience that extends from the desktop to the cloud. The Nimbella Serverless Platform handles configuration of all integral services such as storage management, capacity provisioning, auto-scaling, monitoring, and logging.

The Nimbella Serverless Platform allows developers to create stateful, stateless, streaming or long-running applications that can operate at enterprise scale, with high performance and built-in security. Nimbella's technology is also suited for complex use-cases that employ artificial intelligence and machine learning techniques.

Components of Nimbella Serverless Platform

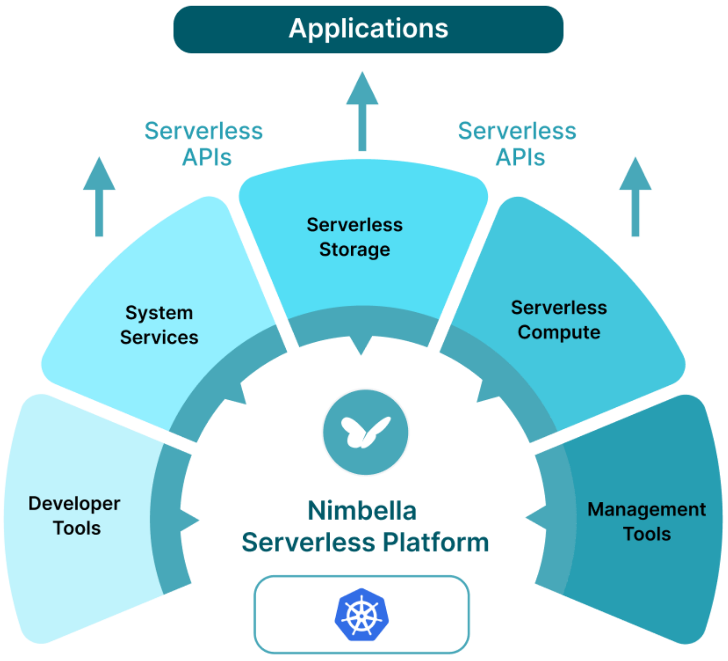
Nimbella Serverless Platform is available as:

- **Nimbella Cloud:** This is managed and hosted by Nimbella and it runs on public cloud infrastructure.
- **Nimbella Enterprise:** This is easily deployable of public and private Kubernetes infrastructure. This allows the enterprises to run in their virtual private cloud or on-premise.

There are five components to Nimbella Serverless platform:

- **Developer Tools:** These are the tools that a developer interfaces to easily build and deploy on Nimbella Cloud or Nimbella Enterprise.
- **Management Tools:** These are the tools used by DevOps team to manage serverless stack on Kubernetes. These tools are only needed for Nimbella Enterprise.
- **Serverless Compute:** This consists of the function engine and container engine for execution of functions and containers.
- **Serverless Storage:** This consists of various forms of integrated storage to support stateful workloads.

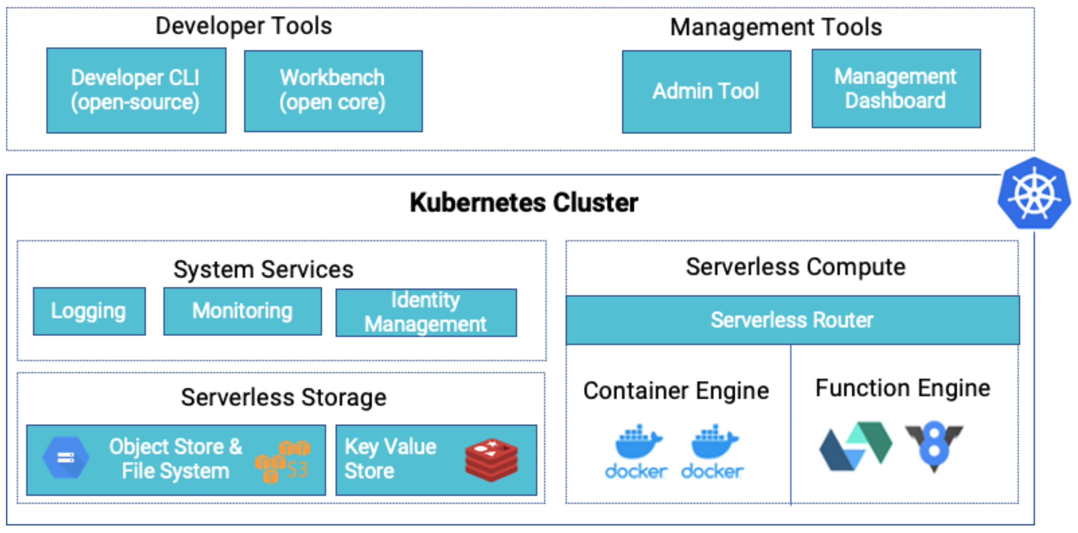
- **System Services:** These are various system services like logging, monitoring and debugging. Other third-party integrations are also part of system services.



Five components of Nimbella Serverless Platform

Nimbella exposes serverless compute, serverless storage and system services as **Serverless APIs** that can be easily consumed in the application.

Platform Architecture



All the components except Developer Tools and Management Tools are deployed in a Kubernetes cluster.

The Serverless Router automatically selects the compute engine – container engine or function engine based on the workload. The Serverless Storage allows the transient state to be stored in key-value store and the static assets are stored in an object store.

The platform is designed to allow upgrades and scaling out of additional resources when the needs of your organization change (from a proof of concept to high throughput and critical services). It is also possible to reduce the number of components deployed, to provide independent testing and staging clusters as well.

Ease of Development

With Nimbella's technology, the developer is afforded a number of technological benefits that accelerate the creation and delivery of applications to the cloud. These benefits include:

- A dedicated and secure domain name for each cloud application.
- Static front-end assets are automatically deployed to and served from a global CDN.
- Back-ends which implement APIs as serverless functions, with resources provisioned on demand, near-instantly. No servers for the developers to manage.
- A secured data bucket to upload files. Limit access as needed.
- Application state recorded in a key-value store, with data accessible to all back-end logic at very low latency.
- Workflows to orchestrate long-running tasks and compose APIs.
- A command line interface and a cloud Workbench to manage Nimbella services, build projects, and deploy applications to the cloud.

The Nimbella platform supports the development of serverless APIs using function in a wide variety of programming languages including:

- Javascript
- TypeScript
- Python
- PHP
- Java
- Go
- Ruby
- Swift

Nimbella provides a uniform programming model for the cloud that allows developers to bring APIs, functions, and containers together in a familiar programming abstraction. Furthermore, a composition engine allows developers to compose services, functions and APIs to satisfy the demands of new and emerging cloud applications. The Nimbella Platform allows organizations to tailor the runtime environments available to functions to security, compliance, and productivity. The Nimbella Platform offers a number of other customizations aimed at addressing the limitations of serverless functions, such as cold starts, memory and timeout limits, and compliance.

Command Line Interface

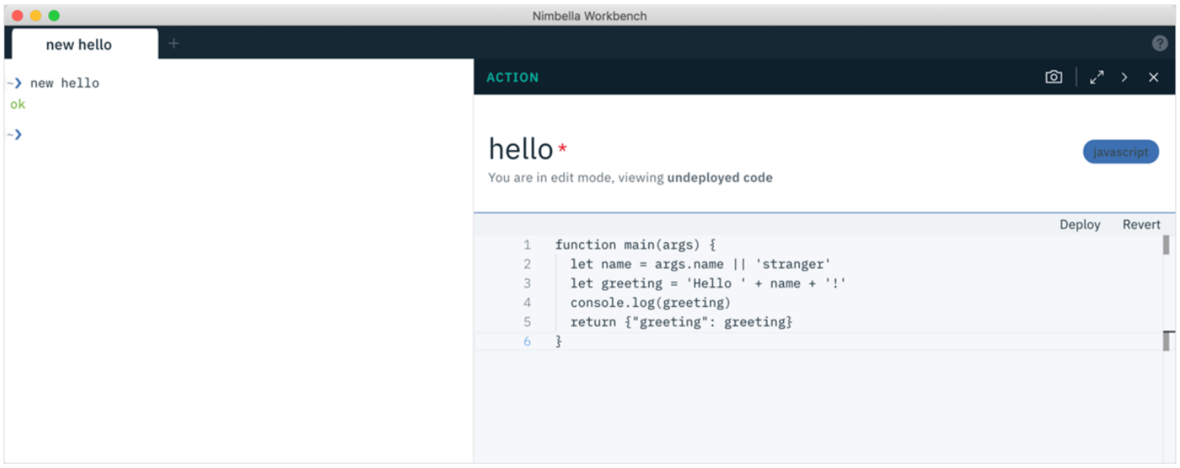
Developers using the Nimbella Platform to build serverless APIs will interact with the Nimbella command line interface (CLI) called `nim`. The CLI runs locally on the developer's terminal and integrates with their typical development workflow. It allows a developer to manage, develop and deploy serverless APIs (even entire applications that include front-end assets, key-value storage, and object storage), from their terminal. The key feature of the CLI is "project deployment" which deploys an entire project, consisting of a set of APIs, to the Nimbella Platform. Some sample CLI commands are shown below:

```
Credential management
nim auth <authkey>

Project commands
nim project create <project name>
nim project deploy <project name>
nim project watch <project name>

Entity management
nim action get --url

Storage management
nim web list
nim object list
nim key-value list
```

The CLI is available as an installable application for macOS, Linux and Microsoft Windows. The CLI is open source and available on GitHub. It may be downloaded from your Nimbella Platform API host.
A project may be deployed from the developer's local file system, or directly from a GitHub repository. Developer documentation for the CLI is available at https://docs.nimbella.com or via the command `nim doc`.

Developer Workbench

In addition to the CLI, Nimbella offers a graphical-user interface as a web application called the Workbench. The Workbench allows developers to use CLI commands (from the terminal) directly in their browser. This allows for one and uniform experience that extends from the desktop to the cloud.



```
~> new hello
ok
~>
```

hello *

You are in edit mode, viewing **undeployed code**

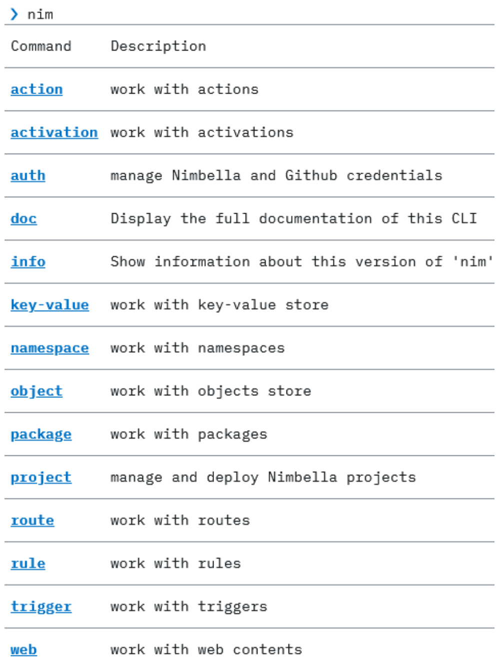
```
function main(args) {
  let name = args.name || 'stranger'
  let greeting = 'Hello ' + name + '!'
  console.log(greeting)
  return {"greeting": greeting}
}
```

The Workbench includes a simple code editor that allows function code to be developed directly in the UI but it is designed to work alongside a developer's IDE such as Visual Studio, Eclipse, IntelliJ.AppCode, Atom, etc. A developer will typically develop and manage source code directories in their IDE and use the Workbench to manage deploying the code, checking logs, managing permissions, etc.

The Workbench allows a developer to manage namespaces, manage packages, manage functions, list function activations, work with an integrated object store, work with an integrated key/value store, manage routes, rules and triggers and manage web contents.

Managing these same resources is also available via the CLI, the main difference between the CLI and Workbench is that the Workbench provides a graphical interface for code editing and graphics related to activation information and function composition.

Some commands may not be available in the Workbench when they require local filesystem access.



```
> nim
```

Command	Description
action	work with actions
activation	work with activations
auth	manage Nimbella and Github credentials
doc	Display the full documentation of this CLI
info	Show information about this version of 'nim'
key-value	work with key-value store
namespace	work with namespaces
object	work with objects store
package	work with packages
project	manage and deploy Nimbella projects
route	work with routes
rule	work with rules
trigger	work with triggers
web	work with web contents

When using the Nimbella public cloud, a developer connects the Workbench to their Platform namespace using an authorization key which can be generated directly from the web browser or from the terminal. This creates a secure connection between the Workbench and the Nimbella Cloud. The same authorization key authorization mechanism is used for private cloud deployment.

Autodeploy

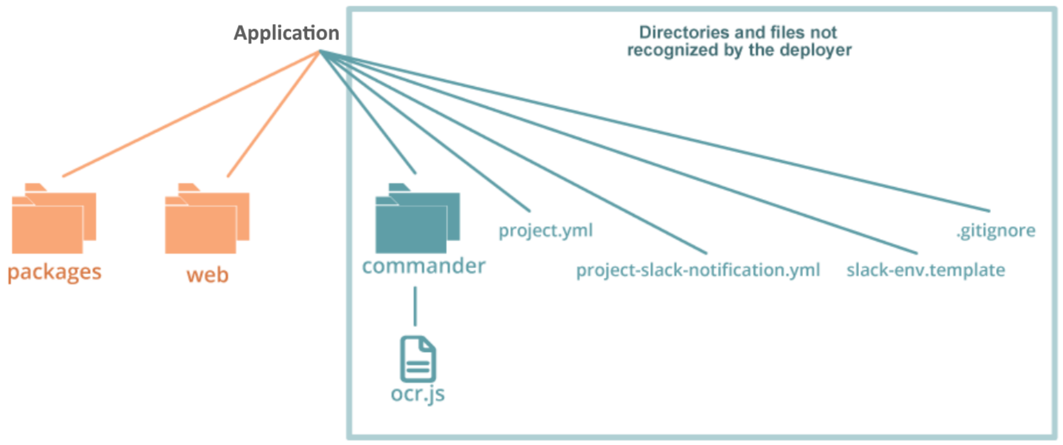
A developer can use the Nimbella CLI to set functions to automatically deploy to the Nimbella cloud when they are changed. The CLI process will watch for file changes and will automatically deploy function changes to the cloud when it detects source code has changed.

This speeds development as developers can use their IDE, save code out and simply test it. Instead of a cycle of ***edit, save, deploy, test*** the interactive development cycle shortens to ***edit, save, test***.

Front-end and back-end code in a single project

The Nimbella Platform allows developers to manage the front end and back end as a single, unified project.

A cloud project organized as a Nimbella project consists of a *packages* directory that contains the back-end logic (i.e., APIs) of the project and a *web* directory that contains the static, front-end web content of the application. The web contents may be built from any popular web front-end framework (e.g., React, Vue). They are deployed globally and served from a content delivery network when using the public Nimbella cloud.


Application
Directories and files not recognized by the deployer
packages
web
commander
ocr.js
project.yml
project-slack-notification.yml
slack-env.template
.gitignore

During development, developers would typically run a local Node server when developing Angular, Node.js or Vue.js front-ends. These locally developed user-interfaces connect to back-end API function code running in the Nimbella cloud. Developers can iteratively develop back-end code, with changes automatically deployed, and iteratively develop front end code, with changes automatically reflected by the local Node.js server.

Various user-interface components would connect independently to various back-end functions. An application with a web-page displaying a React chart and table, for example, may have one back-end API function that returns data to display in the chart and another to display data in the table. Each React, Vue or Angular object can then fetch data at the same time, from the two different functions, allowing them to render simultaneously.

When local development of the user-interface is complete, the user-interface can be built into a single, static set of webpack'ed front-end code that is hosted and delivered by the Nimbella platform. As a result, both the front-end code and back-end code is delivery, hosted, run and managed by the Nimbella platform.

Use your own source code control system

The Nimbella Platform does not contain its own source code management (SCM). Developers are free to use whatever SCM they are used to, including local Git, CVS, Github, or Gitlab to name some.

Postman is typically used to manage and test collections of back-end APIs. Nimbella has a custom integration with Postman that is available to automatically create function source code stubs based on a Postman API definition or OpenAPI.

Ease of Deployment

Unlike other serverless systems, the Nimbella Platform automatically provisions resource for:

- Custom hostnames including SSL certificates.
- An integrated key-value store to maintain state.
- An integrated object store.

A developer can directly make use of these resources from their source code. They are fully managed and secured by the Nimbella Platform.

Nimbella Deployer

The Nimbella Deployer is part of the Nimbella CLI and Nimbella Workbench. It can be used by developers or operations people to deploy a release into a Nimbella cloud, public or private.

As stated previously an entire project including front-end code, back-end code and associated key-value and object store is managed as a unified unit and can be deployed as a single, unified release.

Automatic hostnames with SSL certificate management

Using the Nimbella Platform, developers are relieved from having to create hostnames and managing SSL certificates for their projects. Each function deployed to the cloud is automatically associated with a custom hostname w/SSL certificate and path. SSL certs are automatically kept updated and current by the platform utilizing industry-standard LetsEncrypt certificates.

Private Cloud Deployment

Unlike many other serverless platforms, the Nimbella Serverless Platform can be deployed in a private cloud. The Nimbella Platform itself was developed utilizing a container-based architecture. The Nimbella Platform for a Private Cloud includes the following components.

- API Gateway
- Redis
- Zookeeper
- CouchDB
- OpenWhisk Invoker
- OpenWhisk Controller
- ElasticSearch
- Kafka
- Prometheus
- Grafana

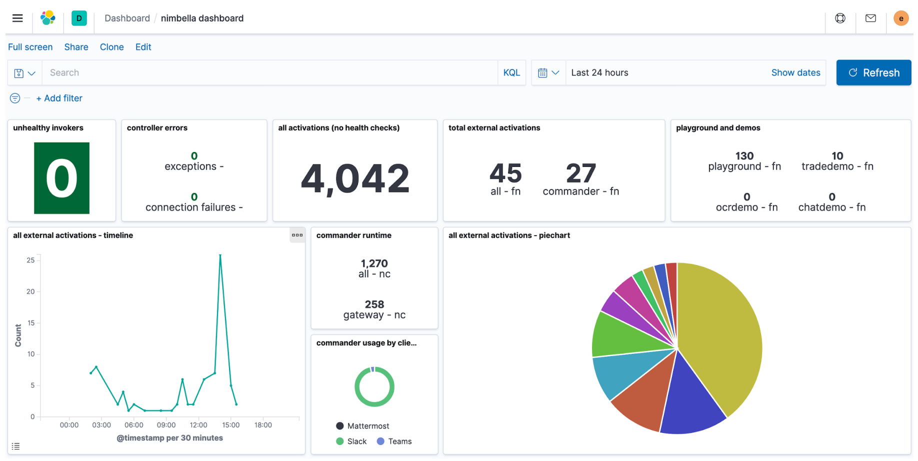
Admin Tool

The entire platform is deployed using the Nimbella Admin Tool (called nimadmin) which automates the entire build and deploy steps for the platform. The tool also provides administrative functions to operate and perform routine maintenance of the platform. Some example automation commands provided by the Nimbella Administration Tool are:

```
Deploy entire stack
nimadmin deploy <system>

Redeploy specific components
nimadmin redeploy <component>

Shell into a specific component
nimadmin shell <component>

Share/add resources
nimadmin namespace modify <id>

search component logs
nimadmin logs --search <component> <expression>
```

Management Dashboard

A Prometheus-based UI is included for monitoring of historical load activity and other statistics about the platform. Additionally, the platform includes Kibana dashboards for monitoring developer usage, and to view system logs.



Kubernetes

The Nimbella Platform runs on top of Kubernetes. The platform consists of several containerized microservices. It is deployed using the Nimbella Administrator Tool which automates all of the deployment and configuration steps needed to install Nimbella on an existing Kubernetes cluster. It is available for macOS and Linux. The tool interacts with the underlying Kubernetes cluster using standard `kubectl` commands.

In addition to the basic components of the Nimbella Serverless Platform, a platform deployment contains containers that run serverless functions. When a function is invoked, the Invoker looks for a suitable container to execute the function. A pool of “stem-cell” initialized containers may be available for the function or a container that previously ran the same function in the same namespace, a “warm” container may be available. If no suitable container is available a “cold” container will be obtained and initialized and the function will be run in it.

Standard kubectl commands are used to monitor and manage the platform containers. A Prometheus-based monitoring UI is available to monitor and show historical load activity and other statistics about the Platform Kubernetes deployment.

Platform deployment

For the private cloud, the deployment is typically “bring your own Kubernetes”. If there is no existing Kubernetes deployment to deploy on top of, Nimbella can aid creating a private Kubernetes cluster to deploy the Nimbella Platform on top of.

To build a test Kubernetes cluster, a Docker-in-Docker approach for running Kubernetes directly on top of Docker can be used. A minimum of 8GB of memory and 4 virtual CPUs is required to deploy the Nimbella Platform.

When building a production cluster, at least 1 worker node with 8GB of memory and 4 virtual CPUs are required. Significantly larger clusters can be deployed by scaling up the replica count of the various components and labeling multiple nodes as invoker nodes.

Serverless Compute

Nimbella Serverless Platform utilizes an extended version of Apache OpenWhisk as its underlying serverless compute foundation. Apache OpenWhisk is the leading open-source platform for serverless computing, with proven enterprise-adoption at IBM, Adobe, Naver, and other large organizations.

It supports both function runtimes as well as container runtimes. Functions are executed internally in the same manner as in Apache OpenWhisk with a NGINX router front-ending the system, internal routing utilizing Kafka to manage the ingress of function requests and responses to and from Docker containers and a custom invoker to manage containers and function invocation.

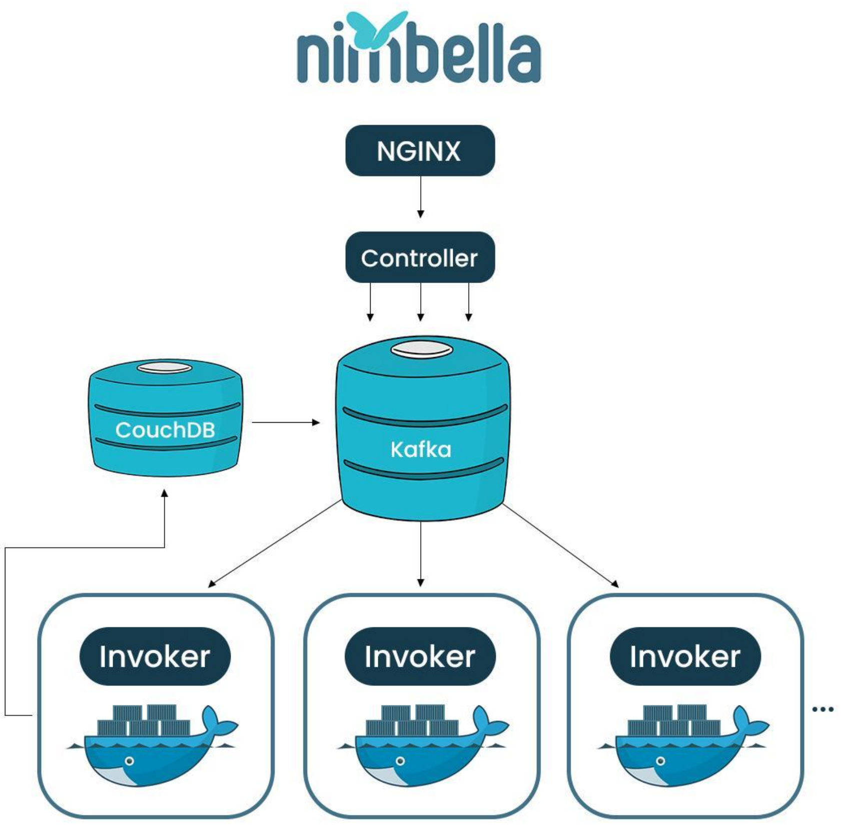
Nimbella extended Apache OpenWhisk by integrating the automated deployment with Kubernetes, adding a customized Workbench, advanced CLI, deployment tools and adding features such as the ability to manage state with an integrated key-value store and integrated object store.

The management tools automate the underlying platform running on Kubernetes and allow any organization to easily support the deployment with admin tools and management dashboard.

Nimbella architecture is extensible to support other execution systems such as Knative, Lambda and other public cloud function services.

The path of function invocation

A function API call from a client first uses DNS to resolve to an IP address. Listening at the IP address is a NGINX server that performs SSL key exchange and also load-balances to a set of back end Controllers.



After being passed the invocation request from NGINX, the Controller examines the local cache for information associated with the function being invoked. A CouchDB instance contains information about each function and it is accessed if information isn't locally available.

The Controller sends a message to a Kafka broker which queues the request if necessary and passes the invocation request to an Invoker. The Invoker is responsible for finding an existing Docker container (warm start) or allocating a new Docker container and initializing it. Not shown in the diagram is ElasticSearch where activation logs are written by the Invoker. After the function is complete, the response is based back to NGINX for delivery back to the client.

Security

End-to-end security is maintained through HTTPS (TLS) transport with SSL certificates specific to a deployment. The Nimbella Platform requires two ingress points. The API host and Admin host. The API host is typically open to internet access but may be restricted to a virtual private network. The Admin host access is restricted to the Kubernetes cluster.

Containers are used to isolate and prevent one function from accessing the information of any other running function. Containers are only reused if the same function and namespace are executed in a container.

This re-use of containers can have its benefits. Globals in code in containers that are reused maintain their values. It is common for a database connection to be opened and stored in a function global. A subsequent invocation of the same function will maintain the global and thus the previous database connection allowing pooling of database connections to be maintained automatically.

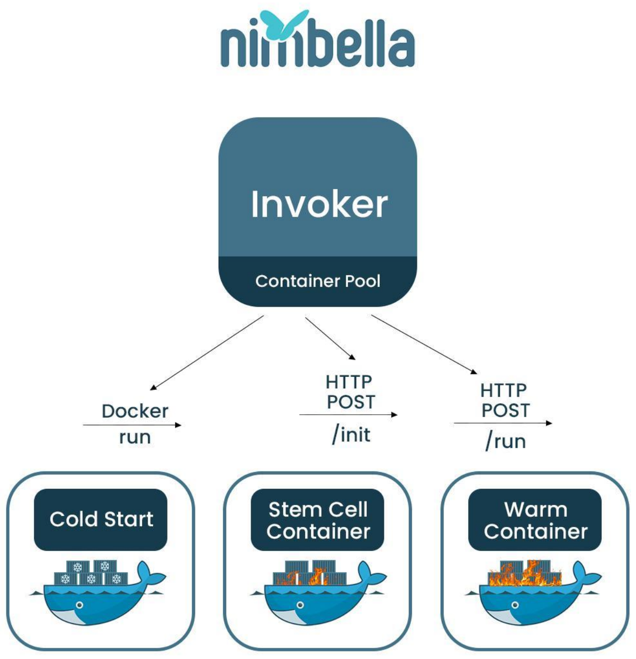
Failover

Load balancing and failover exists on the DNS and HTTP levels with DNS load balancing in the cloud available via usage of Amazon Route53 and load balancing via NGINX. Private deployments of the Nimbella Platform do not need DNS based local balancing, except in extremely special cases. NGINX based load balancing for private clouds is sufficient to maintain high-availability in a private deployment of the platform.

Additionally, for private deployments, VIP based load balancing with common hardware load balancing could be utilized.

Performance

When benchmarked, Apache OpenWhisk has always been one of the fastest, if not the fastest platform for executing serverless functions. Nimbella has tuned the platform for performance in a number of ways. The figure below shows the different ways a serverless API may execute, with decreasing system overhead from left to right.



- **Cold Starts:** A cold start occurs when a serverless API is first invoked. If the code which implements the API has dependencies, such as a Node.js function that requires a number of Node packages, there can be a delay before the API starts executing. The delay is attributed to initialization which must occur to load the required dependencies. Nimbella's project deployment system allows a developer to mark the dependencies an API has. A container image can be built ahead of time (and started) to pre-load the required dependencies. This drastically improves cold-start performance.
Different programming languages have different cold start performance. The Nimbella Platform permits operators to create custom runtime images to suit the needs of their organization.
- **Stem Cells:** The platform creates stem cell containers with speculative container initialization. Stem cells are a pool of ready to execute containers, available for new APIs. Once a stem cell is allocated, the pool of ready containers is regenerated.
- **Warm Starts:** When a container has already been initialized for a given API, the start of execution can be extremely fast. In a distributed deployment, a warm start is typically under 10 milliseconds. In a lean configuration, it can be 4 milliseconds or less.

The following Nimbella Platform optimizations are possible:

- Customizing each of the language runtime environments to bundle dependencies specific to your applications. This customization reduces the deployment time for serverless APIs, and improves cold-start times.
- Wrapping every serverless API with canonical pre and post checks. For example, to integrate with internal logging infrastructure used by your organization, or to audit API responses for leaked secrets.
- Configuring various system limits including:
    - CPU and memory resource limits available to each serverless API execution.
    - Maximum duration an API may execute.
    - Maximum logging output per invocation of an API.
    - Limits controlling API invocations per minute, and maximum in flight activations.
    - The number of concurrent containers allocated on each Invoker pod.

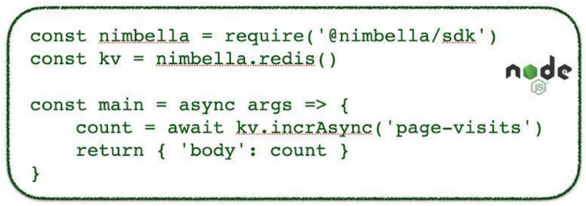
Serverless Storage

State is key to almost all applications. For example, in a web browsing session, web browser cookies are normally associated with server state, allowing a browser cookie key to map to information about a user on the server side.

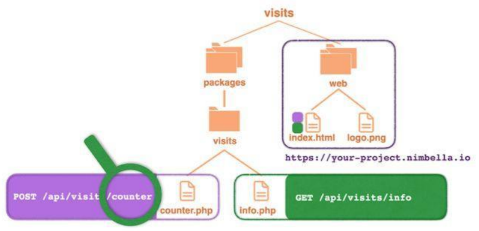
The Nimbella Platform comes with an integrated key/value store that is available to programmers without having to sign up for or create any additional resources.

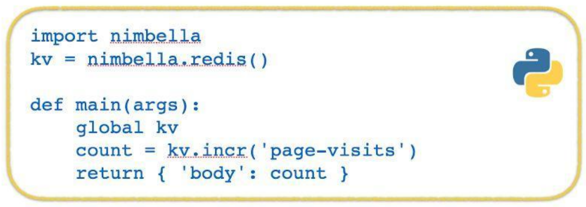
Integrated Key-value store

State is key to almost all APIs and cloud applications. For example, in a web browsing session, web browser cookies are normally associated with server state, allowing a browser cookie to map to information about a user on the server side.
The Nimbella Platform comes with an integrated key-value store that is available to developers without having to explicitly provision additional resources for their APIs.
The key-value store integrated with the Nimbella Platform is implemented using Redis. The Redis server is strictly accessible from within the Kubernetes cluster hosting the Nimbella Platform. The Nimbella Platform automatically injects the Redis host IP, port and access keys into the runtime environment which is executing a serverless API. Each developer is allocated a separate Redis instance. In this way, it is possible for a developer to create multiple personas, one for each of their APIs, to keep the state of the APIs completely segregated.

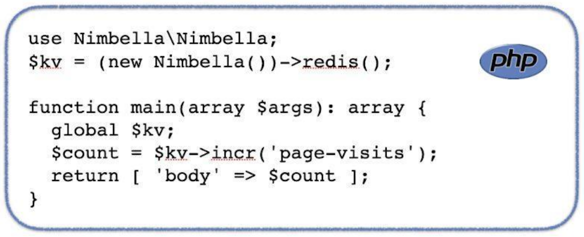


```
const nimbella = require('@nimbella/sdk')
const kv = nimbella.redis()

const main = async args => {
    count = await kv.incrAsync('page-visits')
    return { 'body': count }
}
```





```
import nimbella
kv = nimbella.redis()

def main(args):
    global kv
    count = kv.incr('page-visits')
    return { 'body': count }
```



```
use Nimbella\Nimbella;
$kv = (new Nimbella())->redis();

function main(array $args): array {
  global $kv;
  $count = $kv->incr('page-visits');
  return [ 'body' => $count ];
}
```

No additional work is needed by the developer, or the platform operator, to create or maintain the key-value store. It is maintained by the Nimbella Platform. The developer simply uses the key-value store as shown above.

Integrated object store

Applications generally include icons, images, associated documents, machine learning training data, static files and other media. The Nimbella Platform includes a distributed application store that developers can use to store and access stored files in function code.

There is no need for a developer or operations person to do any work to utilize the integrated Nimbella object store. A programmer can simply add files to their project to have them stored along with deployed functions. In function code, the object store can be accessed programmatically through an API. Every namespace has its own object store to ensure isolation and security between applications in the system.

In its cloud deployment, the Nimbella Platform's object store is actually a convenience layer on top of object storage provided by major cloud providers such as Google, Amazon and Microsoft.

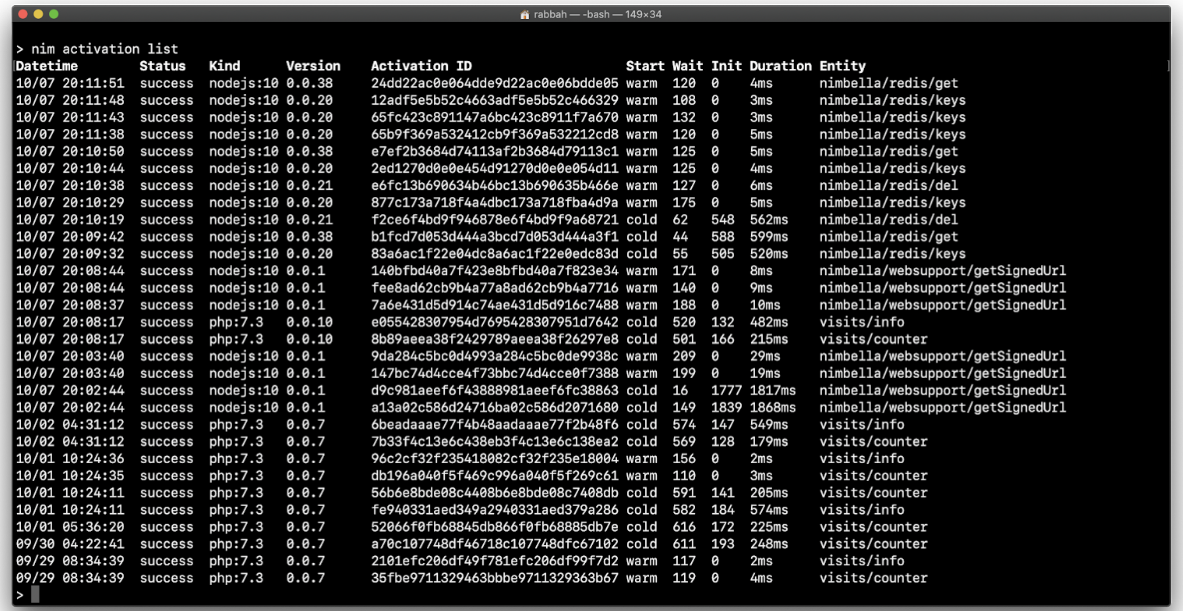
System Services

Logging

A key to developer and operations experience is the ability to debug a running application. To aid in this, the Nimbella Platform has a fully featured logging system that logs the details of every function activation on the serverless platform.

When a function is activated, it generates an activation log record. The record includes information about the activation including the full function input, full output, execution time, type of container utilized and other information. The platform does have the ability to turn off storing full response information about activations on a namespace basis when full responses may contain sensitive data.

Logs can be accessed using the CLI or the Workbench. Logs may also be routed to existing logging services such as Elastic or Splunk. The Workbench also contains a graphical user-interface to activations, including a heatmap showing the most popular activations and any activation errors.



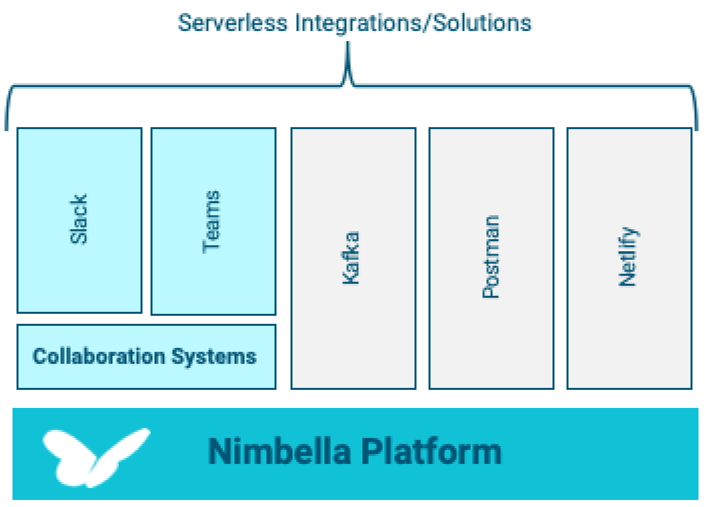
```
> nim activation list
Datetime       Status    Kind      Version    Activation ID                    Start Wait Init Duration Entity
10/07 20:11:51 success   nodejs:10 0.0.38     24dd22ac0e064dde9d22ac0e06bdde05 warm  120  0    4ms      nimbella/redis/get
10/07 20:11:48 success   nodejs:10 0.0.20     12adf5e5b52c4663adf5e5b52c466329 warm  108  0    3ms      nimbella/redis/keys
10/07 20:11:43 success   nodejs:10 0.0.20     65fc423c891147a6bc423c8911f7a670 warm  132  0    3ms      nimbella/redis/keys
10/07 20:11:38 success   nodejs:10 0.0.20     65b9f369a532412cb9f369a532212cd8 warm  120  0    5ms      nimbella/redis/keys
10/07 20:10:50 success   nodejs:10 0.0.38     e7ef2b3684d74113af2b3684d79113c1 warm  125  0    5ms      nimbella/redis/get
10/07 20:10:44 success   nodejs:10 0.0.20     2ed1270d0e0e454d91270d0e0e054d11 warm  125  0    4ms      nimbella/redis/keys
10/07 20:10:38 success   nodejs:10 0.0.21     e6fc13b690634b46bc13b690635b466e warm  127  0    6ms      nimbella/redis/del
10/07 20:10:29 success   nodejs:10 0.0.20     877c173a718f4a4dbc173a718fba4d9a warm  175  0    5ms      nimbella/redis/keys
10/07 20:10:19 success   nodejs:10 0.0.21     f2ce6f4bd9f946878e6f4bd9f9a68721 cold  62   548  562ms    nimbella/redis/del
10/07 20:09:42 success   nodejs:10 0.0.38     b1fcd7d053d444a3bcd7d053d444a3f1 cold  44   588  599ms    nimbella/redis/get
10/07 20:09:32 success   nodejs:10 0.0.20     83a6ac1f22e04dc8a6ac1f22e0edc83d cold  55   505  520ms    nimbella/redis/keys
10/07 20:08:44 success   nodejs:10 0.0.1      140bfbd40a7f423e8bfbd40a7f823e34 warm  171  0    8ms      nimbella/websupport/getSignedUrl
10/07 20:08:44 success   nodejs:10 0.0.1      fee8ad62cb9b4a77a8ad62cb9b4a7716 warm  140  0    9ms      nimbella/websupport/getSignedUrl
10/07 20:08:37 success   nodejs:10 0.0.1      7a6e431d5d914c74ae431d5d916c7488 warm  188  0    10ms     nimbella/websupport/getSignedUrl
10/07 20:08:17 success   php:7.3   0.0.10     e055428307954d7695428307951d7642 cold  520  132  482ms    visits/info
10/07 20:08:17 success   php:7.3   0.0.10     8b89aeea38f2429789aeea38f26297e8 cold  501  166  215ms    visits/counter
10/07 20:03:40 success   nodejs:10 0.0.1      9da284c5bc0d4993a284c5bc0de9938c warm  209  0    29ms     nimbella/websupport/getSignedUrl
10/07 20:03:40 success   nodejs:10 0.0.1      147bc74d4cce4f73bbc74d4cce0f7388 warm  199  0    19ms     nimbella/websupport/getSignedUrl
10/07 20:02:44 success   nodejs:10 0.0.1      d9c981aeef6f43888981aeef6fc38863 cold  16   1777 1817ms   nimbella/websupport/getSignedUrl
10/07 20:02:44 success   nodejs:10 0.0.1      a13a02c586d24716ba02c586d2071680 cold  149  1839 1868ms   nimbella/websupport/getSignedUrl
10/02 04:31:12 success   php:7.3   0.0.7      6beadaaae77f4b48aadaaae77f2b48f6 cold  574  147  549ms    visits/info
10/02 04:31:12 success   php:7.3   0.0.7      7b33f4c13e6c438eb3f4c13e6c138ea2 cold  569  128  179ms    visits/counter
10/01 10:24:36 success   php:7.3   0.0.7      96c2cf32f235418082cf32f235e18004 warm  156  0    2ms      visits/info
10/01 10:24:35 success   php:7.3   0.0.7      db196a040f5f469c996a040f5f269c61 warm  110  0    3ms      visits/counter
10/01 10:24:11 success   php:7.3   0.0.7      56b6e8bde08c4408b6e8bde08c7408db cold  591  141  205ms    visits/counter
10/01 10:24:11 success   php:7.3   0.0.7      fe940331aed349a2940331aed379a286 cold  582  184  574ms    visits/info
10/01 05:36:20 success   php:7.3   0.0.7      52066f0fb68845db866f0fb68885db7e cold  616  172  225ms    visits/counter
09/30 04:22:41 success   php:7.3   0.0.7      a70c107748df46718c107748dfc67102 cold  611  193  248ms    visits/counter
09/29 08:34:39 success   php:7.3   0.0.7      2101efc206df49f781efc206df99f7d2 warm  117  0    2ms      visits/info
09/29 08:34:39 success   php:7.3   0.0.7      35fbe9711329463bbbe9711329363b67 warm  119  0    4ms      visits/counter
> 
```

External Monitoring

The Nimbella Platform integrates Datadog monitoring. This is enabled by adding your Datadog API key to the Nimbella Platform configuration. The Datadog monitoring is used to monitor the health of the API host, system response times, disk and memory utilization.

Integrations

Nimbella has developed applications using its own platform to integrate with a number of third-party services such as Postman, Netlify and several collaboration platforms.



Collaboration/Messaging systems

Nimbella has Nimbella Commander, an entirely serverless application that runs on the Nimbella Platform that allows users to create custom commands in the Slack and Microsoft Teams messaging systems. Additionally, commands created can be invoked from the command line or via API.

Nimbella Commander adds messaging system user access roles, in-messaging logging, secret values and command set management into the most popular messaging systems. It is available as a one-click install from the app stores of the various messaging system providers.

Postman

Nimbella has the Nimbella Postman Integration, which allows a developer to generate source code stubs from a Postman-defined API. Developers can quickly mock up back end functions by defining the back-end API in Postman and then generating stub functions from them that can be immediately deployed and run on the Nimbella Platform.

Kafka

Nimbella has Kafka sink and source connectors to integrate with the Kafka message bus. The Kafka sink connector will invoke a Nimbella function, passing it a set of data from a Kafka topic, allowing Kafka data to be transformed or acted on in Nimbella functions.

The Kafka source connector allows a Nimbella function to easily push data into a Kafka topic. Both connectors can be used with Confluent's hosted Kafka service.

Netlify

The Nimbella add-on provides Netlify developers with a powerful programming model for serverless functions and workflows, and features to build stateful APIs using Nimbella's integrated key-value and object stores.

The plugin enables developers to use many languages for your serverless functions. For example: Python, PHP, Ruby, Rust, Typescript, Deno, Java, and Swift in addition to JavaScript and Golang. It also allows developers to run functions for longer durations, and consume more events, not just HTTP requests.

Demo applications

- Optical Character Recognition (OCR)
- Chat
- Stock Trading
- QR Code Generator

Summary

The Nimbella Serverless Platform is available as a cloud offering (running on public clouds) and also as installable software that can be run on Kubernetes services like AWS EKS, GCP GKE and other Kubernetes distributions running in private cloud or private infrastructure.

Unlike other Function-as-a-Service (FaaS) platforms, Nimbella is not just a FaaS platform, instead it is a true Serverless platform that enables a developer to build applications to handle variety of enterprise workloads which may be stateful or stateless.

What some enterprise developers are saying…

I spent significant amounts of time building collections for deploying APIs to AWS using Lambda, so when I had an API deploying in just a couple of minutes with Nimbella, I was pretty impressed with what they have done to serverless by making it easy, but also bringing in the critical ingredients you need to actually deliver meaningful apps.

With Nimbella, deploying my app was a breeze; all I had to care about was getting the functionality that I needed right and not be worried about the deployment. It helped in channelizing most of my time in building the product that I wanted rather than thinking about how I will have to make it available.

Nimbella really helped me focus on development aspects only and avoiding deployment and cloud complexities. In a completitive world, agility is something that we cannot escape our focus from, where Nimbella comes to our rescue.

About Nimbella

Nimbella Serverless Platform makes serverless development frictionless and portable. It unifies the programming experience and provides the necessary abstractions enabling the developers to code once and run on cloud-of-their-choice. Available as a hosted, managed service and as a full stack solution on any public or private Kubernetes environment, it fully supports an enterprise's multi-cloud strategy. Founded by Silicon Valley infrastructure and open-source veterans Anshu Agarwal, Rodric Rabbah and Eric Swildens, Nimbella was recently named one of CRN's "Top 20 Cloud Infrastructure Companies to watch in 2021". For more information, please visit nimbella.com and follow @nimbella on Twitter.

Download your copy of **The Forrester Wave™: Function-As-A-Service Platforms, Q1 2021** to learn more about this fast-evolving market, and why Nimbella, a multi-cloud stateful serverless platform provider was named as a Strong Performer.

Download Report

Contact Us

Email: info@nimbella.com
Website: www.nimbella.com