

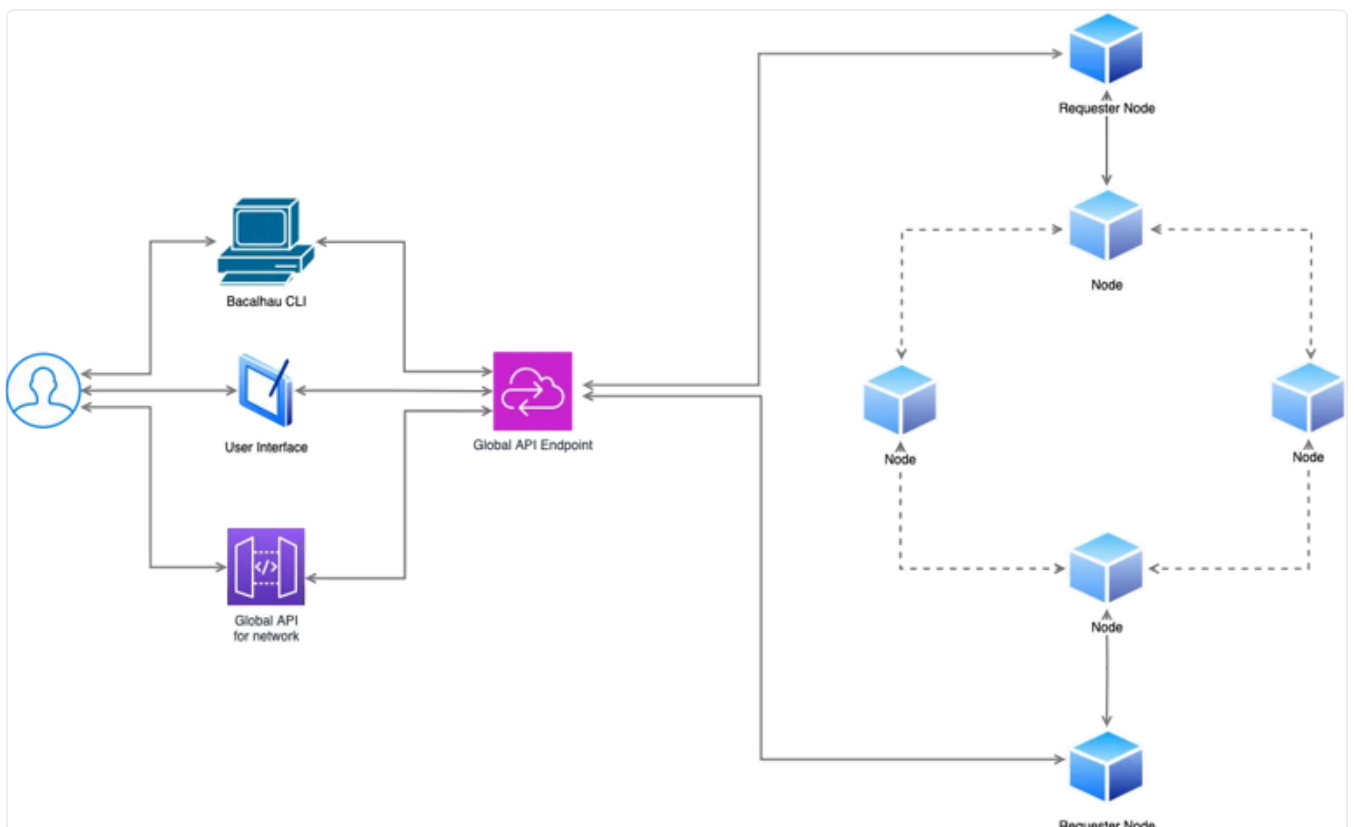


# How Bacalhau Works

In this tutorial we will go over the components and the architecture of Bacalhau. You will learn how it is built, what components are used, how you could interact and how you could use Bacalhau.

## Chapter 1 - Architecture

Bacalhau is a peer-to-peer network of nodes that enables decentralized communication between computers. The network consists of two types of nodes, which can communicate with each other.



Bacalhau Architecture

The requester and compute nodes together form a p2p network and use gossiping to discover each other, share information about node capabilities, available resources and health status. Bacalhau is a peer-to-peer network of nodes that enables decentralized communication between computers.

**i Requester Node:** responsible for handling user requests, discovering and ranking compute nodes, forwarding jobs to compute nodes, and monitoring the job lifecycle.

**Compute Node:** responsible for executing jobs and producing results. Different compute nodes can be used for different types of jobs, depending on their capabilities and resources.

To interact with the Bacalhau network, users can use the Bacalhau CLI (command-line interface) to send requests to a requester node in the network. These requests are sent using the JSON format over HTTP, a widely-used protocol for transmitting data over the internet. Bacalhau's architecture involves two main sections which are the **core components** and **interfaces**.

> Components overview

## Core Components

The core components are responsible for handling requests and connecting different nodes. The network includes two different components:

> Requester node

> Compute node

## Interfaces

The interfaces handle the distribution, execution, storage and publishing of jobs. In the following all the different components are described and their respective protocols are shown.

> Transport

> Executor

> Storage Provider

> Publisher

## Chapter 2 - Job cycle

### Job preparation

You can create jobs in the Bacalhau network using various [job types](#) introduced in version 1.2. Each job may need specific variables, resource requirements and data details that are described in the [Job Specification](#).

> Advanced job preparation

### Job Submission

You should use the Bacalhau client to send a task to the network. The client transmits the job information to the Bacalhau network via established protocols and interfaces. Jobs submitted via the Bacalhau CLI are forwarded to a Bacalhau network node at <http://bootstrap.production.bacalhau.org/> via port `1234` by default. This Bacalhau node will act as the requester node for the duration of the job lifecycle.

Bacalhau provides an interface to interact with the server via a REST API. Bacalhau uses 127.0.0.1 as the localhost and 1234 as the port by default.

CLI   API

```
bacalhau job run [flags]
```

You can use the command with [appropriate flags](#) to create a job in Bacalhau using JSON and YAML formats.

You can use the `bacalhau docker run` [command](#) to start a job in a Docker container. Below, you can see an excerpt of the commands:

> Bacalhau Docker CLI commands

You can also use the `bacalhau wasm run` [command](#) to run a job compiled into the (WASM) format. Below, you can find an excerpt of the commands in the Bacalhau CLI:

> Bacalhau WASM CLI commands

## Job Acceptance

When a job is submitted to a requester node, it selects compute nodes that are capable and suitable to execute the job, and communicate with them directly. The compute node has a collection of named executors, storage sources, and publishers, and it will choose the most appropriate ones based on the job specifications.

## Job execution

The selected compute node receives the job and starts its execution inside a container. The container can use different executors to work with the data and perform the necessary actions. A job can use the docker executor, WASM executor or a library storage volumes. Use [Docker Engine Specification](#) to view the parameters to configure the Docker Engine. If you want tasks to be executed in a WebAssembly environment, pay attention to [WebAssembly Engine Specification](#).

## Results publishing

When the Compute node completes the job, it publishes the results to **S3's remote storage, IPFS**.

Bacalhau's seamless integration with IPFS ensures that users have a decentralized option for publishing their task results, enhancing accessibility and resilience while

reducing dependence on a single point of failure. View [IPFS Publisher Specification](#) to get the detailed information.

Bacalhau's S3 Publisher provides users with a secure and efficient method to publish task results to any S3-compatible storage service. This publisher supports not just AWS S3, but other S3-compatible services offered by cloud providers like Google Cloud Storage and Azure Blob Storage, as well as open-source options like MinIO. View [S3 Publisher Specification](#) to get the detailed information.

## Chapter 3 - Returning Information

The Bacalhau client receives updates on the task execution status and results. A user can access the results and manage tasks through the command line interface.

### Get Job Results

To Get the results of a job you can run the following command.

```
bacalhau job get [id] [flags]
```

One can choose from a wide range of flags, from which a few are shown below.

#### Usage:

```
bacalhau job get [id] [flags]
```

#### Flags:

<code>--download-timeout-secs</code> duration	Timeout duration for IPFS downloads.
<code>-h, --help</code>	help for get
<code>--ipfs-connect</code> string	The ipfs host multiaddress to connect
<code>--ipfs-serve-path</code> string	path local Ipfs node will persist dat
<code>--ipfs-swarm-addr</code> strings	IPFS multiaddress to connect the in-p
<code>--ipfs-swarm-key</code> string	Optional IPFS swarm key required to c
<code>--output-dir</code> string	Directory to write the output to.
<code>--private-internal-ipfs</code>	Whether the in-process IPFS node shou
<code>--raw</code>	Download raw result CIDs instead of n

### Describe a Job

To describe a specific job, inserting the ID to the CLI or API gives back an overview of the job.

CLI API

```
bacalhau job describe [id] [flags]
```

You can use the command with [appropriate flags](#) to get a full description of a job in yaml format.

## List of Jobs

If you run more than one job or you want to find a specific job ID

CLI API

```
bacalhau job list [flags]
```

You can use the command with [appropriate flags](#) to list jobs on the network in yaml format.

## Job Executions

To list executions follow the following commands.

CLI API

```
bacalhau job executions [id] [flags]
```

You can use the command with [appropriate flags](#) to list all executions associated with a job, identified by its ID, in yaml format.

# Chapter 4 - Monitoring and Management

The Bacalhau client provides the user with tools to monitor and manage the execution of jobs. You can get information about status, progress and decide on next steps. View the [Bacalhau Agent APIs](#) if you want to know the node's health, capabilities, and deployed

Bacalhau version. To get information about the status and characteristics of the nodes in the cluster use [Nodes API Documentation](#).

## Stop a Job

CLI   API

```
bacalhau job stop [id] [flags]
```

You can use the command with [appropriate flags](#) to cancel a job that was previously submitted and stop it running if it has not yet completed.

## Job History

CLI   API


```
bacalhau job history [id] [flags]
```

You can use the command with [appropriate flags](#) to enumerate the historical events related to a job, identified by its ID.

## Job Logs

```
bacalhau job logs [flags] [id]
```

You can use this [command](#) to retrieve the log output (stdout, and stderr) from a job. If the job is still running it is possible to follow the logs after the previously generated logs are retrieved.

 To familiarize yourself with all the commands used in Bacalhau, please view [CLI Commands](#)

Previous  
Welcome to the Bacalhau Docs

Next  
Installation

Last updated 20 days ago

