Rust applications on Deta.Space & Shuttle.rs

Shuttle.rs and **Deta.Space** are platforms for development and deployment of cloud applications. They both support **Rust**. Let's look at which one you may want.
You can see the slides **online** (with a dark theme, too). If you **clone** or **download** them and follow a few steps in its README.md, you can also render the slides locally and offline. Of course, your comments or pull requests are welcome.

☰

Disclaimers (part 1/2)

- This is **not** a complete introduction to either platform. Their existing documentation is already awesome.
- Both are agile/work in progress. Especially, functionality can be ahead of the documentation. Join and shape it!
- Not fully comparable. They share or differ in some features, but other features or limitations are unique to one or the other.
- Updated in mid August 2023.

Disclaimers (part 2/2)

- Most features and limits considered here are **not** Rust-specific. However, they affect which platform to choose based on/for
    - an existing application or system design, or
    - a new system design, or
    - integration with non-Rust components, or
    - portable or platform-specific API's/database/storage, or
    - sharing of applications
    - developer base & instance owner base
    - data isolation, or
    - limits/quotas and affected use cases, or
    - options for cloud providers, or private cloud, or
    - an (optional) commercial use.

☰

Overview

- Brief comparison of Deta.Space and Shuttle.rs at code level. For more details see:
  - Shuttle.rs:
    - **sys-info.shuttleapp.rs** & its **source** using Axum
    - **http-headers.shuttleapp.rs** and its **source** using Actix-web
  - Deta.Space:
    - **sysinfo-1-s4498989.deta.app** & its **source** using Axum
    - **tmpwdav-1-q0047082.deta.app** & its **source** using Warp (which is based on Hyper)
- Features and Quantitative differences between them.
- Let's see which platform may suit you.

Spoiler Alert

I love each. So exciting.

☰

Deta.Space: Spacefile with defaults (GLIBC)


```
! Spacefile-glibc
  1     # Spacefile Docs: https://go.deta.dev/docs/spacefile/v0
  2     v: 0
  3   ∨ micros:
  4   ∨   - name: main
  5         primary: true
  6         public: true
  7         src: .
  8         engine: custom
  9         dev: cargo run
 10   ∨     commands:
 11           - cargo build --release
 12         run: target/release/tmp-wdav-deta-space
 13   ∨     include:
 14           - target/release/tmp-wdav-deta-space
 15
```

Deta.Space: Spacefile with MUSL target

```
Spacefile - tmp-wdav-deta-space - Visual Studio Code

  util.rs        Cargo.toml      ! Spacefile      main.rs   M

! Spacefile
  1    # Spacefile Docs: https://go.deta.dev/docs/spacefile/v0
  2    v: 0
  3    micros:
  4      - name: main
  5        primary: true
  6        public: true
  7        src: .
  8        engine: custom
  9        dev: cargo run
 10        commands:
 11          - rustup target add x86_64-unknown-linux-musl
 12          - cargo build --release --target=x86_64-unknown-linux-musl
 13        run: target/x86_64-unknown-linux-musl/release/tmp-wdav-deta-space
 14        include:
 15          - target/x86_64-unknown-linux-musl/release/tmp-wdav-deta-space
```

Deta.Space: Cargo.toml

```rust
use dav_server::{fakels::FakeLs, localfs::LocalFs, DavHandler};
use std::net::{IpAddr, SocketAddr};
use std::{env, fs, io};
use warp::Filter;

const WDAV_SYMLINKS: &'static str = "/tmp/wdav_symlinks";

#[tokio::main]
async fn main() -> io::Result<()> {
    let port: String = env::var("PORT").unwrap_or("8080".to_string());
    let port: u16 = port.parse::<u16>().unwrap();

    let cd: PathBuf = std::env::current_dir().unwrap();
    let ip: IpAddr = "127.0.0.1".parse().unwrap();
    let addr: SocketAddr = SocketAddr::new(ip, port);
```
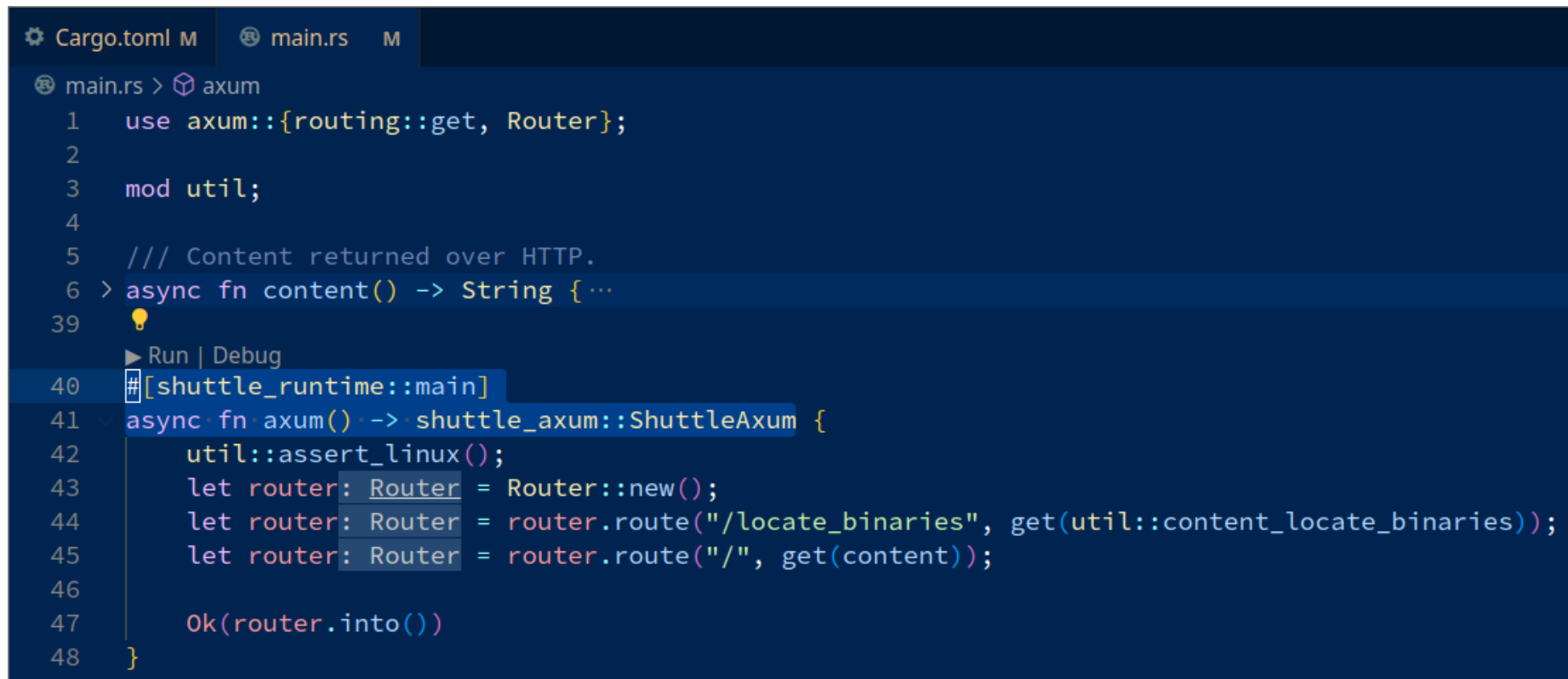
Deta.Space: Getting the port number

```rust
util.rs        Cargo.toml        Spacefile        Spacefile-glibc        main.rs    M

main.rs > main
1    use dav_server::{fakels::FakeLs, localfs::LocalFs, DavHandler};
2    use std::net::{IpAddr, SocketAddr};
3    use std::{env, fs, io};
4    use warp::Filter;
5
6    const WDAV_SYMLINKS: &'static str = "/tmp/wdav_symlinks";
7
8    #[tokio::main]
   ▶ Run | Debug
9    async fn main() -> io::Result<()> {
10       let port: String = env::var("PORT").unwrap_or("8080".to_string());
11       let port: u16 = port.parse::<u16>().unwrap();
12
13       let cd: PathBuf = std::env::current_dir().unwrap();
14       let ip: IpAddr = "127.0.0.1".parse().unwrap();
15       let addr: SocketAddr = SocketAddr::new(ip, port);
```

Shuttle.rs: Cargo.toml for Axum

```toml
Cargo.toml M        main.rs   M

Cargo.toml > {} dependencies
 1  ∨ [package]
 2    name = "sys-info"
 3    version = "0.1.0"
 4    edition = "2021"
 5    description = "System info & limits of free tier on Shuttle.rs"
 6    homepage = "https://sys-info.shuttleapp.rs"
 7    repository = "https://github.com/peter-kehl/sys-info.shuttleapp.rs"
 8    keywords = ["async", "cloud", "cloud-native", "shuttle", "shuttle-service"]
 9    license = "MIT OR Apache-2.0"
10
11    # See more keys and their definitions at https://doc.rust-lang.org/cargo/re
12
13  ∨ [dependencies]
14    axum = "0.6.19"
15    shuttle-axum = "0.21.0"
16    shuttle-runtime = "0.21.0"
17    tokio = "1.29.1"
```

Shuttle.rs: Custom crates for Axum

```rust
use axum::{routing::get, Router};

mod util;

/// Content returned over HTTP.
async fn content() -> String { ···

    💡
    ▶ Run | Debug
#[shuttle_runtime::main]
async fn axum() -> shuttle_axum::ShuttleAxum {
    util::assert_linux();
    let router: Router = Router::new();
    let router: Router = router.route("/locate_binaries", get(util::content_locate_binaries));
    let router: Router = router.route("/", get(content));

    Ok(router.into())
}
```

Shuttle.rs: Cargo.toml for Actix-web
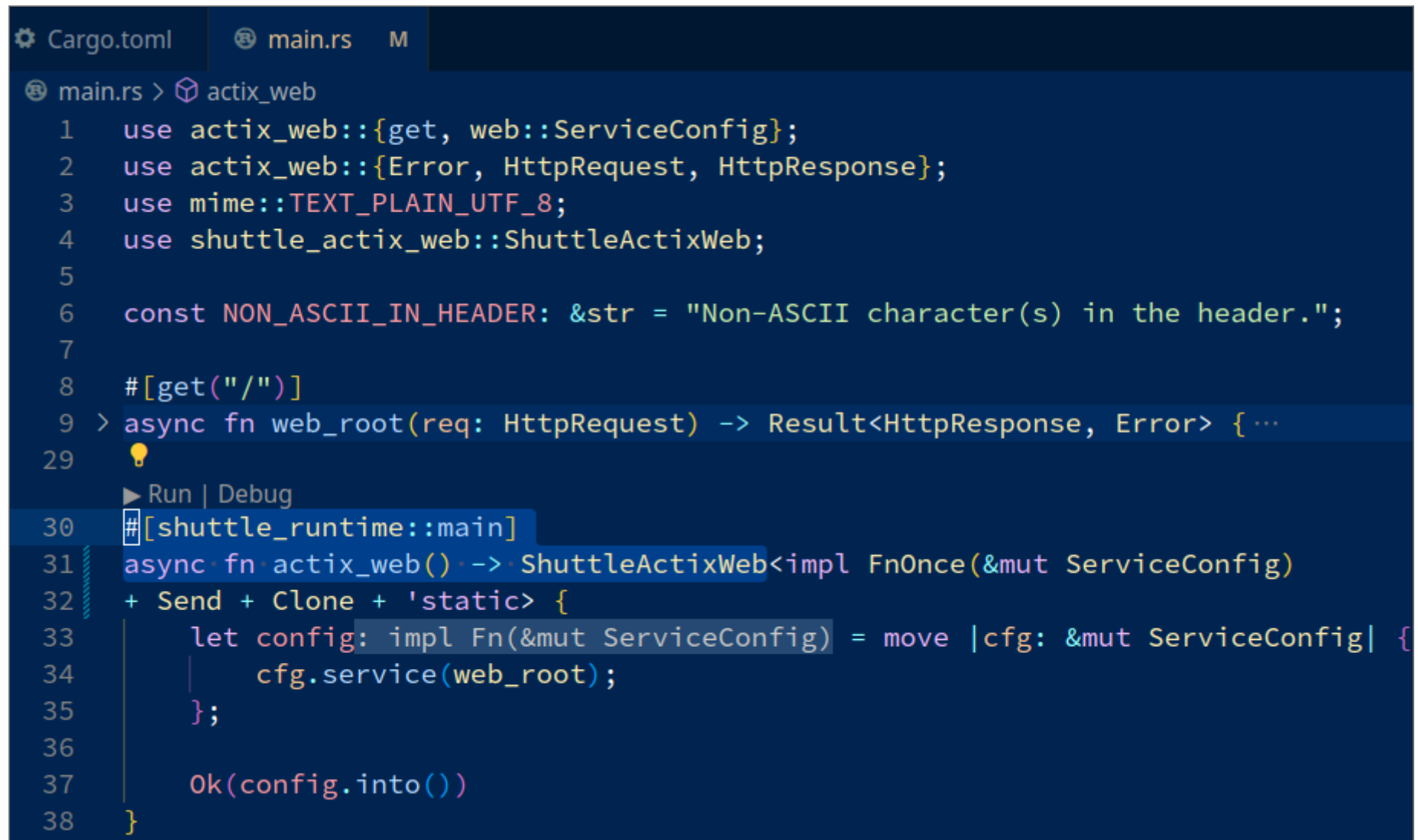
```toml
Cargo.toml > {} dependencies
[package]
name = "http-headers"
version = "0.1.0"
edition = "2021"

[dependencies]
actix-web = "4.3.1"
mime = "0.3.17"
shuttle-actix-web = "0.24.0"
shuttle-runtime = "0.24.0"
tokio = "1.26.0"
```

Shuttle.rs: Custom crates for Actix-web

```rust
Cargo.toml        main.rs    M

main.rs > actix_web
 1   use actix_web::{get, web::ServiceConfig};
 2   use actix_web::{Error, HttpRequest, HttpResponse};
 3   use mime::TEXT_PLAIN_UTF_8;
 4   use shuttle_actix_web::ShuttleActixWeb;
 5
 6   const NON_ASCII_IN_HEADER: &str = "Non-ASCII character(s) in the header.";
 7
 8   #[get("/")]
 9 > async fn web_root(req: HttpRequest) -> Result<HttpResponse, Error> {...
29     💡
     ► Run | Debug
30   #[shuttle_runtime::main]
31   async fn actix_web() -> ShuttleActixWeb<impl FnOnce(&mut ServiceConfig)
32   + Send + Clone + 'static> {
33       let config: impl Fn(&mut ServiceConfig) = move |cfg: &mut ServiceConfig| {
34           cfg.service(web_root);
35       };
36
37       Ok(config.into())
38   }
```

Shared Features

- local development on Linux, Mac OS & Windows
- easy deployment on the respective platform's cloud, in a Docker-like Linux container
- no way to run `su`/`sudo` (or not documented), nor to customize at Docker level (for example: to access remote storage by mounting FUSE file systems)
- No rate-limiting logic or other IP-dependant logic can be performed by the app itself. That is [documented for Deta.Space) (**https://deta.space/docs/en/build/reference/runtime#important-notes-for-micros**). That's also **the reality for Shuttle.rs** (even though not documented). Why? We can't access the client IP (unless you **use Cloudflare DNS**, or any DNS/reverse proxy that injects the client IP in a custom HTTP header).
- support & community on their Discord servers

☰

Deta.Space Features (part 1/4)

- the deployed application can be private (for the developer only), or public ("published")
- Even if the application is public, the developer can deploy an unpublished (test) version of that public application. Such a version is visible only to her/him.
- a public (published) application can still have parts which are private
- to access a private application, or private URLs of a public application, the application owner is authenticated by Deta.Space. (Deta authenticates the user through AWS, but it **doesn't have access to her/his password**.)
- mesh design
  - Mesh of computes: An application can consist of up to five **"micros"** (computes). Each can be developed in any of the supported languages.
  - Suitable if we want to add access control on top of/in front of an existing/3rd party codebase (installed as a part of your application). We can create a "proxy" that performs access control. It then forwards the request to the other (existing/3rd party) application instance, deployed in another micro. (Such a micro would not be public.) We can proxy for example with **warp_reverse_proxy**.

≡

Deta.Space Features (part 2/4)

- mesh design (continued)
  - Mesh of languages/frameworks: Each micro within the same application can use any of the supported languages/frameworks.
- Rust applications don't get special handling. Instead, Rust micros have the "custom" type.
- **Rust bindings for Deta API** are only unofficial.
- No special Rust crates/macros or code, other than getting the basic configuration. That can make the code a little bit more portable/flexible. But then you couldn't store data with Deta.Base/Deta.Store (see below).
- **not with PostgreSQL/MySQL... unless you use a pool manager**
- no restrictions on Rust version, nor on crate versions
- Rust support is new. There are only a few examples of Rust applications so far. But they are growing!
- If you get `GLIBC` issues with Rust on Deta.Space deployment, use **MUSL target** instead.
- **not for: background/long tasks, Discord bots, Websockets** (specifically: not for Discord bots)

≡

Deta.Space Features (part 3/4)

- database & storage provided by the platform is only through their own NoSQL (Deta.Base) and their own storage (Deta.Store) API. If you use those, the source code is not portable. (Unless you create traits or wrappers. Such abstractions are a part of good design. But they add complexity when creating them, and even more so when maintaining.)
- data isolation: if using Deta.Base or Deta.Store, this data is separate per instance owner - even if you clone someone else's published Deta application
- data provisioning: automatic
- `/tmp` (and seemingly **`/dev/shm`, too**)
- CRON-like **scheduled actions**
- subdomain anonymization promotes/suggests using each instance only by its owner. If the application is for public, the users can "fork" their own instances.

≡

Deta.Space Features (part 4/4)

- Instance owners don't need developer skills. In other words, it's easy to have your own/separate deployment (an isolated clone) of an application that someone else published on Deta.Space.
- App Marketplace & commercial model: App Marketplace promotes sharing free applications. Deta.Space is also planning an option of applications to be paid so they would generate revenue for the developer.

☰

Shuttle.rs Features (part 1/3)

- Specializing only in Rust. In addition to hosting, storage and deployment, Shuttle integrates with **multiple Rust web frameworks**. It also provides tutorials on how to connect the **middleware**, and various aspects of **security** and **cryptography**.
- Suitable for background/long tasks (for example: for Discord bots). See "No cold-start and can even have long-running threads" in **FAQ** > "How does this differ from using serverless framework with Rust Lambda and provided runtime?" Also suitable for **Websockets**.
- richer storage
  - wider variety
  - both public/free standards (portable) and proprietary (not portable)
  - RDS (SQL) and handling of migrations/updates
    - Postgres (either a **shared server**, or a **dedicated instance**)
    - MySQL (a **dedicated instance**)
    - MariaDB (a **dedicated instance**)
    - **Turso** (**distributed SQLite** fork, **SQLite-compatible & with 1st class Rust support**). This is currently NOT hosted by Shuttle.rs, but it may be so in the future. Either way, it has a **dedicated crate from Shuttle**.

☰

Shuttle.rs Features (part 2/3)

- richer storage (continued)
  - noSQL: **MongoDB through a shared database**
  - key/value: proprietary **Shuttle Persist**
- data isolation
- A fixed Rust version (currently `1.70`). See **FAQ** > "Which version of Rust...". Similarly, **Turso is pinned to version `0.30.1`**.

☰

Shuttle.rs Features (part 3/3)

- Longer build times: Custom crates and attribute (procedural) macros make the (initial) local build times much longer (than with Deta.Space). Depending on which of the supported Rust framework you choose, your project has initial 300-600 dependencies (in total, most of them being indirect dependencies). It's unclear if Shuttle.rs deployments use (or could use) incremental builds.
- **no `/tmp`**; only `/dev/shm`
- Not promoting/targeting sharing (clones) of applications. Of course, developers are free to publish their code (on GIT or similar) so that others could deploy it on Shuttle.rs, too.
- Instance owners need developer skills, such as running `cargo shuttle ....` In other words, if you want your own/separate deployment of an application that someone else published (on GIT...), it's more work than on Deta.Space.
- Commercial model: For users with more than 5? applications. But, the limits are not enforced yet. And, if you **become a Shuttle.rs hero**, it's free for life!
- use your **own AWS account**

☰

Quantitative & other differences (part 1/2)

Property/Limit	Shuttle.rs	Deta.Space
application/compute size	unspecified	**250MB** per each of up to 5 micros
execution timeout	unspecified	**20s**
RAM per execution	unclear	**250MB**
RAM per container	**6GB** (4GB during high contention: very rare, see **FAQ** > "What happens when I create a project?")	unclear
`/dev/shm` and/or `/tmp`	unspecified (64MB on `/dev/shm`, but **no `/tmp`**)	**512MB** shared between **`/dev/shm` and `/tmp`**

≡

Quantitative & other differences (part 2/2)

Property/Limit	Shuttle.rs	Deta.Space
processes/threads	**4 threads (per project)**	**1024 (per micro)**
HTTP payload	unspecified	**5.5MB**
database and/or object storage	**10GB on free tier**, but not enforced yet. (Plus, free for life if you become a hero.)	unspecified (but personal use is free for life)
clouds/regions	AWS **eu-west** and more planned. (See **FAQ** > Do we plan to support multiple regions?)	**AWS** and potentially planning for **GCP and other clouds**

≡

Choosing between them (part 1/6)

Feature	Deta.Space	Shuttle.rs
	no stars to ***	no stars to ***
`/tmp`	*** (and `/dev/shm`, too)	`/dev/shm` only
max. processes/threads	4 (per app)	1024 (per micro)
max. HTTP payload	max. 5.5MB	
max. RAM (unclear)	250MB	4GB (usually 6GB)
option to use your own AWS account		***
private apps (authenticated through the platform)	*** (one user only: the app instance owner)	

☰

Choosing between them (part 2/6)

Feature	Deta.Space	Shuttle.rs
private parts of public apps (authenticated through the platform)	*** (one user only: the app instance owner)	
single user (owner) authentication	***	(see multi user below)
multi user authentication		not provided, but they have a **detailed tutorial** on how you can implement/integrate it

≡

Choosing between them (part 3/6)

Feature	Deta.Space	Shuttle.rs
mesh of computes	*** (up to 5 computes per app)	(No specific support. You'd need to integrate them into one Rust application. If those parts use different web frameworks and can't be migrated into one framework, you'd need to run those respective frameworks at different ports and proxy/forward to them.)

☰

Choosing between them (part 4/6)

Feature	Deta.Space	Shuttle.rs
mesh of languages/frameworks	***	
1st class support for/primarily for/dedicated to Rust	* (not yet; but Deta is considering moving the internal tooling from Golang to Rust)	*** Rust only; custom crates and macros for config automation/integration; even the tooling is in Rust: **`cargo shuttle`**
background/long tasks/Discord bots	(specifically: no)	***
CRON-like scheduled actions	*** (granularity down to one minute)	

☰

Choosing between them (part 5/6)

Feature	Deta.Space	Shuttle.rs
Stability + Egonomics: Official Rust SDK/bindings	unofficial only	***
Portability > **no** need for special crates/macros	***	
Portability > **no** restrictions on Rust version or channel, or crate versions	***	
Storage > PostgreSQL/MySQL/MariaDB	(no specific support; you need a connection pool)	*** (supported & hosted)
Storage > SQLite/Turso	* officially read-only SQLite only; but Turso is likely to work	*** Turso

≡

Choosing between them (part 6/6)

Feature	Deta.Space	Shuttle.rs
Storage > MongoDB		***
Storage > Proprietary NoSQL	***	
Storage > Proprietary key-value	***	***
Free plan: Any (legal) use	***	*** (with 10GB storage)
Commercial plan: Over the quotas		*** planned
Free model: Marketplace of (personal isolated clones of) applications	***	
Commercial model: Marketplace of paid (personal isolated clones of) applications	*** planned	

≡

Summary

- Choose based on your needs for storage, memory, parallelization, non-Rust/multiple Rust componenet integration, and application owner/user base models.
- Otherwise dig deeper.
- Thank you to both. Loving them. Let's get on cloud, Rustaceans!

☰