**Matthew Planchard** **https://mplanchard.com**
Burlington, VT inbox@mplanchard.com

Principal engineer with extensive experience driving organizational outcomes, building complex systems, and defining high-ownership engineering cultures. Most recently a founding engineer, creating a realtime, inline system handling enterprise-scale traffic with 99.99% uptime and sub-millisecond processing overhead.

## Work History

### Spec (2021–Present)

Senior (2021) | Staff (2023) | Principal Engineer (2025)

As the first application engineer at Spec, I led the implementation of our core platform and its supporting data layer, built our founding engineering team, defined the company engineering culture, and was a primary decision-maker on virtually all major engineering decisions.

Our platform eliminated over 90% of ATO attacks, card testing, and other fraudulent activity for our customers. By analyzing realtime journey data, we could detect, label, and act on anomalous behavior, agentic browsing, and other malicious patterns, giving our customers unparalleled visibility and protection.

#### Core Platform

The core platform is the foundation of Spec. It performs realtime introspection of customer HTTP exchanges, applies customer-defined rules, and persists labeled data for later decisioning and analysis. With our other founding engineers, I designed the data flow, components, and architecture of this system, which would ultimately become core infrastructure for companies with tens of thousands of employees and hundreds of millions of dollars in revenue.

The main components were a proxy server, a workflow engine, and an action runner, all written in Rust. I led the team that owned these components and ensured we met our stringent requirements: average inline processing times below 100 $\mu$s, 99.99% uptime, and a failure rate of less than one in a billion requests.

Core platform capabilities determined the capabilities of the company writ large, so the platform received constant updates and modifications. I ensured feature requests from throughout the organization were implemented without compromising the performance or stability of the platform, which required extensive collaboration with the product team. I also proposed and implemented many features, including several (such as dynamic fallback) critical for landing large, risk-averse enterprise contracts.

#### OLTP Data Layer

We persisted data in heavily optimized PostgreSQL databases, with up to 10k inserts and 50k reads per second for larger customers. Our data layer enabled our industry-leading collation of disparate HTTP exchanges into sessions and further association of those sessions by common entities. Every request destined for our largest customers had access to tens of terabytes of historical data, and we were able to to quickly assess that data for use in workflows.

I designed the schema and rearchitected it several times as we scaled. We aimed for P99 latency less than 10ms for potentially inline queries. I built a variety of performance engineering tools, including an experimental framework that collected statistics on alternative flows in prod environments and a benchmarking suite that collected both timing data and per-process system metrics, particularly CPU, memory, and disk I/O.

#### OLAP Pipeline

When our OLTP-oriented database could no longer support the analytical queries we needed to show value to our customers and drive our own analyses of the efficacy of our platform, I proposed, planned, and implemented a data pipeline, using change data capture to populate data in Confluent Kafka topics, along with a high-performance Rust consumer to pull data into a ClickHouse analytics store.

This OLAP layer gave us the bandwidth and flexibility to write an entirely new frontend experience, enabling configurable dashboards to immediately show the value we were driving for our customers, while also providing a flexible query interface to dig deeper into the data we collected and collated for them.

#### Culture Building

The early engineers and I curated culture with intention. I outlined an Engineering Philosophy document early on, highlighting documentation, autonomy, and learning as key pillars. We wrote and maintained language/framework guidelines, shared planning documents, and talk pages. We instituted weekly pairings, lunch and learns, and formal talks. We gave new engineers support in owning projects from start to finish. We encouraged independent learning with 20% time and quarterly innovation weeks. As a result, we had unusually low turnover, with only two engineers quitting over five years.

**Reproducible Developer and Production Environments**

Engineers were free to work on Windows, Mac, or Linux and to use whatever code editor(s) they chose. To enable this, I created a cross-platform (x86/ARM Linux/Mac/WSL) developer environment using Nix to declaratively define and install all development dependencies, avoiding container virtualization overhead and allowing developer machines to easily run our full suite of production services.

We used the same Nix derivations for our production and CI containers, ensuring local environments and deployed environments were always in sync and reproducible. This eliminated "works on my machine" issues and ensured we could reproduce the vast majority of bugs in deployed environments on our own boxes.

### Bestow (2019–2021)

Senior II (2019) | Staff Engineer (2020)

**Data-Driven Applications**

Reduced time-to-market for new insurance products by a factor of three while improving compliance and reliability by implementing a data-driven application flow, encoding question requirements and validation in JSON documents. This allowed insurance experts to craft the documents and check them for compliance before passing them off to engineering, and allowed frontend and backend teams to write generic presentational and validation logic once, rather than writing custom code for each new insurance product. To address incompatibilities between Python and JS JsonLogic libraries, I developed an open source Rust library with bindings to both languages.

**TypeScript SDK**

Proposed and designed a key component of our white labeling and B2B sales strategy: a TypeScript SDK that consumed our JSON application documents and provided a simple interface with access to active questions, application flow, answer validation, authentication, and submission.

### Previous Experience

**Duo/Cisco (2018–2019)** — **Engineer II**

Worked on native applications and an associated Python server, to provide information to Duo's Zero Trust solution enabling access policies based on device security posture.

**Enthought (2017–2018)** — **Fullstack Web Developer**

Developed IaC deployment pipeline (Docker, Terraform, & Swarm) for both internal and onsite deployments of Enthought's core Python package server, as well as contributing server features.

**ihiji (2014–2017)** — **Intern (2014) | Linux Developer (2015) | Lead (2016)**

Created API server (Python) and monitoring framework (Python/Perl) for IoT devices, utilizing MongoDB, React, and React Native.

## Selected Open Source

| | | |
|---|---|---|
| **cuid-rs** | GitHub | *De facto* standard Rust crate for Collision-resistant Unique IDs (CUIDs) |
| **procfarm** | SourceHut | Rust CLI tool for unix background command parallelization and reporting |

## Education

**M.S. in Biochemistry**, University of Southern Mississippi (2014)
**Dual B.S. in Biology and Chemistry**, University of Southern Mississippi (2012)

## Skills

| | |
|---|---|
| **Leadership** | technical strategy, alignment, architecture, mentoring, documentation, culture |
| **Performance** | profiling, query optimization, benchmarking, observability |
| **Backend** | Async Rust, HTTP, Networking (L4 & L7), C++, C, Go |
| **Scripting** | Shell, Python, Lisp |
| **Data** | PostgreSQL, CDC, ClickHouse, Kafka, Debezium, Redis, MongoDB |
| **Infra** | NixOS, Nix, Docker, Nginx, AWS, Kubernetes, Terraform, CI/CD |