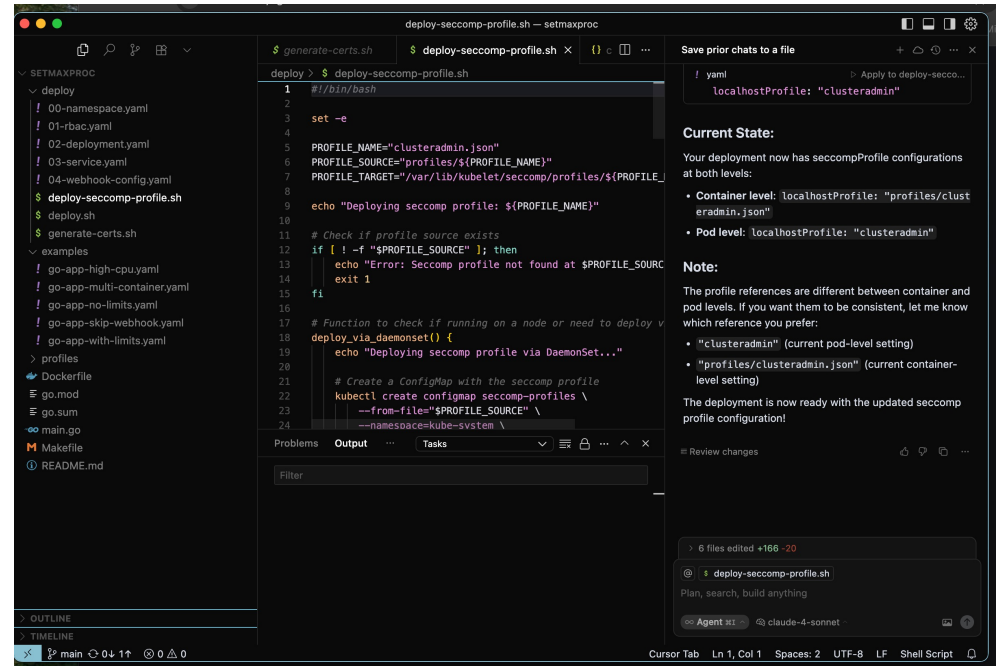# GOMAXPROC via Cursor

Scaling and Setting GOMAXPROC, leveraging Cursor AI
Presented by: Glenn West (gwest@reedhat.com)
Date: August 2025

What is Cursor?

• Cursor is an AI-powered code editor based on VS Code

• Integrates advanced AI into the development workflow

• Powered by OpenAI's Codex/GPT models

• Enhances developer productivity through intelligent assistance

Key Features of Cursor

🤖 AI pair programming with in-editor suggestions

💬 Built-in chat for code explanation & generation

🔧 Context-aware refactoring capabilities

🐛 Live debugging assistance

🚀 GitHub Copilot alternative with enhanced features

Cursor: Advantages & Considerations

Advantages:

🚀 Accelerated development with AI assistance

💬 Real-time code explanations

🔁 Intelligent refactoring suggestions

🔌 Compatible with VS Code extensions

🔍 Advanced search and symbol resolution

Considerations:

📶 Requires internet connectivity for AI features

🔒 Data privacy and security implications

💸 Subscription cost for premium features

🧠 Risk of over-dependence on AI assistance

Ideal Use Cases for Cursor

🏗️ Rapid prototyping and MVP development

📚 Learning new codebases or programming languages

⚡ Generating boilerplate and template code

🔍 Complex debugging and troubleshooting

👥 Solo developers and small agile teams

🎯 Time-sensitive development projects

What are Mutating Webhooks?

🔧 Part of Kubernetes admission controllers

✏️ Modify (mutate) objects during create/update operations

⏰ Execute before objects are persisted to etcd

🛡️ Enable automated policy enforcement and defaults

🔄 Provide dynamic configuration capabilities

Allows the webhook to "overwrite" parameters on the fly!!!

Mutating Webhook Process Flow

1. User submits request to Kubernetes API server

2. Request enters the admission controller pipeline

3. Mutating webhook intercepts the incoming request

4. Webhook analyzes and modifies the object

5. Modified object is returned to API server

6. Final object is validated and persisted to etcd

Mutating Webhook Use Cases

🔌 Auto-injecting sidecars (Istio, Envoy)

🏷️ Adding default labels and annotations

📊 Setting resource limits and requests

🛡️ Enforcing security policies at runtime

🔧 Namespace and RBAC configuration

📝 Compliance and governance automation

In our example: Adjusting golang MAX Proc

Mutating Webhooks: Benefits & Challenges

Benefits:

✅ Automated consistency and policy enforcement

✅ Reduced manual intervention and errors

✅ Flexible and powerful operational control

✅ Dynamic configuration capabilities

Challenges:

❌ Potential API performance impact if misconfigured

❌ Complex debugging and troubleshooting

❌ Increased operational complexity

❌ Dependency on webhook availability

GoMaxProcs and Openshift

GOMAXPROCS is an environment variable and a function within Go's runtime package that controls the maximum number of operating system threads that can simultaneously execute user-level Go code.

By default, OpenShift Container Platform masters and nodes utilize all available CPU cores on the system they are running on. This behavior is influenced by the Go runtime's GOMAXPROCS environment variable.

In more recent versions of Go (and thus OpenShift components built with Go), GOMAXPROCS defaults to the number of available cores. This means that if GOMAXPROCS is not explicitly set, Go programs will, by default, utilize all detected CPU cores for concurrent execution.

GoMaxProcs and AMD

AMD CPU's now have a huge number of cores. 192 Cores and 384 Threads.

   AMD EPYC 5th Gen – Model 9965

The CPU supports dual sockets, allowing 384 Cores and 768 Threads

Thus the default in a GoLang based application is 768 Threads!!!!!!

Baremetal Defaults Vs Container

- For a bare metal golang application, setting this to default to physical cores/threads makes perfect sense
- In a container environment, not so much
- Developer can set GOMAXPROCS aligned with containers CPU limit
- Use a library like uber-go/automaxprocs
- Set resource limits in your application manifest

Developers forget~

- Customer case – Perfomance issues, where kubelet was taking huge amounts of CPU time
- Determined that multiple factors contributed:
  - AMD EPYC Processor (Didn't show up on intel)
  - Application test creating lots of kubelet resource thrashing.
  - GOMAXPROC set to infinity and beyond.
  - Kubernetes garbage collection using all the threads.

Quick Proof Of Concept

- While kubelet is started by Systemd any other golang based application could hit same problem.

- A more tenable default value that is container oriented would be way of avoiding future issues.

SetMaxProc

- This webhook app helps optimize Go applications running in containers by automatically setting appropriate GOMAXPROCS values. It prevents Go applications from creating too many OS threads when they don't have access to all the host's CPUs, which can lead to poor performance and increased context switching.

Features

- **Automatic Detection**: Identifies Go applications based on container images and environment variables

- **Smart Calculation**: Sets GOMAXPROCS based on CPU limits/requests

- **Configurable**: Skip webhook for specific pods using annotations

- **Secure**: Uses TLS certificates for secure communication

How It Works

- The webhook intercepts pod creation/update requests
- Identifies containers that appear to be Go applications
- Calculates appropriate GOMAXPROCS value based on CPU resources:
  - Uses CPU limits if available
  - Falls back to CPU requests if no limits are set
  - For containers without resource constraints: uses max(system_cpus / max_pods, 2) where max_pods = 250
  - Rounds fractional CPU values up to the nearest integer
  - Minimum value is 1
- Adds the GOMAXPROCS environment variable to the container

Default Calculation Logic

- For containers without CPU limits or requests, the webhook uses an intelligent default:
- GOMAXPROCS = max(system_cpu_count / max_pods_per_node, 2)
- Where max_pods_per_node = 250 (typical Kubernetes node limit).
- This approach:
- **Conservative resource allocation**: Assumes maximum pod density to prevent over-allocation
- **Ensures minimum performance**: Guarantees at least 2 processes for reasonable concurrency
- **Node-aware scaling**: Considers the realistic maximum workload per node
- **Prevents resource exhaustion**: Avoids setting excessively high GOMAXPROCS on large systems

How it was done

- Decided that the quickest universal method to implement this was a webhook. (Yes, I hate them too.)
- Used the Cursor "Studio"
- Via "Chat" defined what I wanted including the scaling factor and that I wanted a webhook for openshift.

What I got:

- Really good doc, and scaling table.
- Ask for it to add some more test values to make sure logic was working correctly.
- Test cases in example
- Deployment configs and script

Source of project

https://github.com/glennswest/setmaxproc

The other side of this

- The "Cursor" studio is a poor implementation of Microsoft Visual Studio Code.
  - Should be a plugin like everything else.
- History – In ai, there is a need to save the "history" of what you ask, to be able to replicate it. In Cursor this is in a sqlite database, that you will need 3rd party tools to extract.

Additional Features needed:

- Implementation as a plugin.
- Implementation to save history in format that is readable, and reusable, that can be part of version control.
  - Including doc in readme or a link

AI Whispering

- Random Implementation
  - Knowing how to phrase your question is important. This is your only control to what you will get.
  - What version it supports Is totally undefined.
    - This is a common issue across AI's, where there is no concept of version ie openshift version included in model. Some are getting better, with a comment to the effect, but still this is a blocker.
  - AI assumptions may totally be different that your expectations.

New Concept

- Debugging gets to be more interesting
- Traditional model of stubbing, and working thru each section of code goes away, its "woosh" and your left to find your way thru your completed project.

Conclusion

- Great Tool along the way
- Still rough around the edges
- Huge productivity increase – 2 weeks to 2 hours for development
- Debug/Test may have gotten worse, still not enough time to debug it.
- Far Better doc
- Need to polish further AI whispering.

Code:

- https://github.com/glennswest/setmaxproc