✂️ AutoPrune

A Serverless Cloud Cost Optimization Tool

AutoPrune is a full-stack serverless application designed to detect and delete unused AWS EBS volumes ("Zombie Volumes") that silently drain your budget. It features a high-performance **Go** backend for scanning resources and a modern **Next.js** dashboard for visualization.

📋 Project Overview

The Problem

When EC2 instances are terminated, their attached EBS volumes (hard drives) often remain behind. These "Zombie Volumes" sit in an `available` state, storing obsolete data while charging you monthly storage fees ($0.08 - $0.10 per GB).

The Solution

AutoPrune scans your AWS account for these unattached volumes, calculates the potential monthly savings, and provides a "One-Click Prune" interface to delete them.

🛠️ Tech Stack & Requirements

Component	Technology	Reasoning
Infrastructure	**SST (v3)**	Deploys the entire stack (Infra as Code) with one command.
Backend	**Go (Golang)**	Compiles to a tiny binary for near-instant Lambda "cold starts."
Frontend	**Next.js (React)**	Professional, responsive dashboard UI.
Cloud	**AWS Lambda**	Serverless compute. Costs $0 when not used.
Security	**IAM Roles**	Least-privilege access to manage EC2 volumes.

Prerequisites

Ensure you have the following installed on your machine:

1. **Node.js (v18+)**
2. **Go (v1.20+)** (Run `go version` to verify)
3. **AWS** CLI (Configured with `aws configure`)

📂 Project Structure

Your project folder (`autoprune/`) must look exactly like this:

```
autoprune/
├── sst.config.ts          # Infrastructure definition
├── package.json           # Node dependencies
├── app/                   # Next.js Frontend
│   ├── page.tsx           # Dashboard UI
│   └── layout.tsx         # Layout wrapper
└── functions/             # Go Backend
    ├── go.mod             # Go module definition (Generated)
    ├── go.sum             # Go dependencies (Generated)
    └── hunter.go          # The Logic Script
```

🚀 Step-by-Step Implementation Guide

Phase 1: Initialize the Project

If you haven't created the folder yet, run these commands in your terminal:

```
mkdir autoprune
cd autoprune
mkdir app functions
```

Phase 2: The Infrastructure (`sst.config.ts`)

Create a file named `sst.config.ts` in the root folder. This tells AWS how to wire everything together.

```
/// <reference path="./.sst/platform/config.d.ts" />

export default $config({
  app(input) {
    return {
      name: "autoprune",
      home: "aws",
      providers: {
        aws: { region: "us-east-1" }
      }
    };
  },
  async run() {
    // 1. Create the Backend (Go Lambda)
    const api = new sst.aws.Function("AutoPruneApi", {
      handler: "functions/hunter.go", // This points to your Go file
      runtime: "go",                  // Tells SST to compile it
      url: true,                      // Creates a public HTTP endpoint
      permissions: [
        {
          actions: ["ec2:DescribeVolumes", "ec2:DeleteVolume"],
          resources: ["*"]
        }
      ],
```

```
    });

    // 2. Create the Frontend (Next.js)
    new sst.aws.Nextjs("AutoPruneWeb", {
      link: [api], // Automatically passes the API URL to the frontend
    });
  },
});
```

Phase 3: The Backend Logic (Go)

This is the step you were missing. We need to create the Go script and install the AWS SDK.

1. **Initialize the Go Module:**

```
cd functions
go mod init autoprune

# Install AWS SDK for Go v2
go get [github.com/aws/aws-lambda-go/lambda](https://github.com/aws/aws-lambda-go/l
go get [github.com/aws/aws-lambda-go/events](https://github.com/aws/aws-lambda-go/e
go get [github.com/aws/aws-sdk-go-v2](https://github.com/aws/aws-sdk-go-v2)
go get [github.com/aws/aws-sdk-go-v2/config](https://github.com/aws/aws-sdk-go-v2/c
go get [github.com/aws/aws-sdk-go-v2/service/ec2](https://github.com/aws/aws-sdk-go
```

2. **Create the Script:** Create a file named `hunter.go` inside the `functions/` folder.

```
package main

import (
        "context"
        "encoding/json"
        "fmt"
        "log"

        "[github.com/aws/aws-lambda-go/events](https://github.com/aws/aws-lambda-go/ev
        "[github.com/aws/aws-lambda-go/lambda](https://github.com/aws/aws-lambda-go/la
        "[github.com/aws/aws-sdk-go-v2/aws](https://github.com/aws/aws-sdk-go-v2/aws)"
        "[github.com/aws/aws-sdk-go-v2/config](https://github.com/aws/aws-sdk-go-v2/co
        "[github.com/aws/aws-sdk-go-v2/service/ec2](https://github.com/aws/aws-sdk-go-
        "[github.com/aws/aws-sdk-go-v2/service/ec2/types](https://github.com/aws/aws-s
)

// Data structure for our response
type ZombieVolume struct {
        VolumeId      string  `json:"VolumeId"`
        Size          int32   `json:"Size"`
        PricePerMonth float64 `json:"PricePerMonth"`
        State         string  `json:"State"`
}

type RequestBody struct {
        Action string `json:"action"`
```

```
}

func HandleRequest(ctx context.Context, request events.APIGatewayProxyRequest) (events
        // 1. Load AWS Configuration
        cfg, err := config.LoadDefaultConfig(ctx)
        if err != nil {
                return errorResponse("Failed to load AWS config: " + err.Error()), nil
        }
        client := ec2.NewFromConfig(cfg)

        // 2. Parse User Action (Scan vs Delete)
        var reqBody RequestBody
        if request.Body != "" {
                json.Unmarshal([]byte(request.Body), &reqBody)
        }
        action := reqBody.Action
        if action == "" {
                action = "scan"
        }

        // 3. Scan for "Available" (Zombie) volumes
        input := &ec2.DescribeVolumesInput{
                Filters: []types.Filter{
                        {
                                Name:   aws.String("status"),
                                Values: []string{"available"},
                        },
                },
        }

        result, err := client.DescribeVolumes(ctx, input)
        if err != nil {
                return errorResponse("Failed to fetch volumes: " + err.Error()), nil
        }

        var zombies []ZombieVolume
        deletedCount := 0

        // 4. Process Volumes
        for _, v := range result.Volumes {
                z := ZombieVolume{
                        VolumeId:      *v.VolumeId,
                        Size:          *v.Size,
                        PricePerMonth: float64(*v.Size) * 0.08, // Approx $0.08/GB
                        State:         string(v.State),
                }

                if action == "delete" {
                        // --- DELETION LOGIC ---
                        _, err := client.DeleteVolume(ctx, &ec2.DeleteVolumeInput{
                                VolumeId: v.VolumeId,
                        })
                        if err == nil {
                                deletedCount++
                                // Only add to list if successfully deleted or scanned
                                zombies = append(zombies, z)
                        } else {
                                log.Printf("Failed to delete %s: %v", *v.VolumeId, err
                        }
```

```
                } else {
                        // Just Scanning
                        zombies = append(zombies, z)
                }
        }

        // 5. Create Response
        message := fmt.Sprintf("Scan complete. Found %d zombies.", len(zombies))
        if action == "delete" {
                message = fmt.Sprintf("Prune complete. Deleted %d volumes.", deletedCo
        }

        responseBody, _ := json.Marshal(map[string]interface{}{
                "message": message,
                "zombies": zombies,
        })

        return events.APIGatewayProxyResponse{
                StatusCode: 200,
                Headers: map[string]string{
                        "Content-Type": "application/json",
                },
                Body: string(responseBody),
        }, nil
}

func errorResponse(msg string) events.APIGatewayProxyResponse {
        return events.APIGatewayProxyResponse{
                StatusCode: 500,
                Body:       fmt.Sprintf(`{"error": "%s"}`, msg),
        }
}

func main() {
        lambda.Start(HandleRequest)
}
```

Phase 4: The Frontend (`app/page.tsx`)

Navigate back to the root (`cd ..`) and then into `app/` . Create `page.tsx` .

```
"use client";
import { useState } from "react";
import { Resource } from "sst"; // This magically imports the Lambda URL

export default function Home() {
  const [zombies, setZombies] = useState<any[]>([]);
  const [loading, setLoading] = useState(false);
  const [statusMsg, setStatusMsg] = useState("");

  async function callApi(action: "scan" | "delete") {
    setLoading(true);
    try {
      // Resource.AutoPruneApi.url comes from SST
      const res = await fetch(Resource.AutoPruneApi.url, {
        method: "POST",
```

```
      body: JSON.stringify({ action: action })
    });
    const data = await res.json();
    setZombies(data.zombies || []);
    setStatusMsg(data.message);
  } catch (e) {
    setStatusMsg("Error connecting to AutoPrune API");
  }
  setLoading(false);
}

return (
  <main className="min-h-screen bg-black text-white p-12 font-sans">
    <div className="max-w-3xl mx-auto">
      <header className="mb-10 flex justify-between items-center border-b border-gra
        <div>
          <h1 className="text-4xl font-bold text-emerald-400">✂️ AutoPrune</h1>
          <p className="text-gray-400 mt-2">Cloud Cost Optimization & Zombie Hunter<
        </div>
        <button
          onClick={() => callApi("scan")}
          disabled={loading}
          className="px-6 py-2 bg-gray-800 hover:bg-gray-700 rounded-lg font-medium
        >
          {loading ? "Scanning..." : "🔍 Scan Now"}
        </button>
      </header>

      {statusMsg && (
        <div className="mb-8 p-4 bg-gray-900 border border-gray-700 rounded text-eme
          {statusMsg}
        </div>
      )}

      <div className="space-y-4">
        {zombies.length === 0 && !loading && (
          <div className="text-center py-20 text-gray-500">
            No zombies detected. Your cloud is clean!
          </div>
        )}

        {zombies.map((z) => (
          <div key={z.VolumeId} className="flex justify-between items-center p-5 bg-
            <div>
              <p className="font-mono text-lg text-emerald-200">{z.VolumeId}</p>
              <p className="text-sm text-gray-500">{z.Size} GB • {z.State}</p>
            </div>
            <div className="text-right">
              <p className="text-xl font-bold text-white">${z.PricePerMonth.toFixed(
              <p className="text-xs text-gray-500">per month</p>
            </div>
          </div>
        ))}
      </div>

      {zombies.length > 0 && (
        <div className="mt-10 border-t border-gray-800 pt-6 text-right">
          <p className="text-gray-400 mb-4 text-sm">
            Ready to save <b>${zombies.reduce((acc, z) => acc + z.PricePerMonth, 0).
```

```
            </p>
            <button
              onClick={() => callApi("delete")}
              disabled={loading}
              className="px-8 py-3 bg-red-600 hover:bg-red-700 text-white font-bold ro
            >
              🔥 Prune All Volumes
            </button>
          </div>
        )}
      </div>
    </main>
  );
}
```

Phase 5: Dependency Install

If you haven't created the `package.json` for the root, do this:

1. Run `npm init -y` in the root folder.
2. Run `npm install sst next react react-dom .`
3. Ensure your `package.json` has the scripts:

```
"scripts": {
  "dev": "sst dev",
  "deploy": "sst deploy"
}
```

Phase 6: Deploy

Now that the `functions/hunter.go` file actually exists, run:

```
npx sst
```